CSRI SUMMER PROCEEDINGS 2020

CSRI Summer Program
Sandia National Laboratories

Editors:
Ahmad A. Rushdi and Michael L. Parks
Sandia National Laboratories

November 1, 2020

SAND#: SAND2020-12580 R

Issued by Sandia National Laboratories, operated for the United States Department of Energy by National Technology & Engineering Solutions of Sandia, LLC.

Available to DOE and DOE contractors from

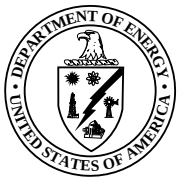
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone:	(865) 576-8401
Facsimile:	(865) 576-5728
E-Mail:	reports@adonis.osti.gov
Online ordering:	http://www.doe.gov/bridge

Available to the public from

U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone:	(800) 553-6847
Facsimile:	(703) 605-6900
E-Mail:	orders@ntis.fedworld.gov
Online ordering:	http://www.ntis.gov/ordering.htm



Preface

The **Computer Science Research Institute (CSRI)** brings university faculty and students to Sandia for focused collaborative research on Department of Energy (DOE) computer and computational science problems. The institute provides an opportunity for university researchers to learn about problems in computer and computational science at DOE laboratories. Participants conduct leading-edge research, interact with scientists and engineers at the laboratories, and help transfer results of their research to programs at the labs. Some specific CSRI research interest areas are: scalable solvers, optimization, adaptivity and mesh refinement, graph-based, discrete, and combinatorial algorithms, uncertainty estimation, mesh generation, dynamic load-balancing, virus and other malicious-code defense, visualization, scalable cluster computers, data-intensive computing, environments for scalable computing, parallel input/output, advanced architectures, and theoretical computer science. The CSRI Summer Program is organized by CSRI and typically includes the organization of a weekly seminar series and the publication of a summer proccedings.

1. CSRI Summer Program 2020. In 2020, the CSRI summer program was executed completely virtually; all student interns worked from home, due to the COVID-19 pandemic. It involved students from 1400—the Center for Computing Research (CCR), 8700—the Center for Homeland Security & Defense Systems, 1300—the Radiation & Electrical Science Center, and 1800—the Material, Physical, and Chemical Sciences Center have actively participated, along with their mentors, in different program activities. We relied heavily on online portals such as SharePoint, MatterMost, Slack, and standard mailing lists to maintain ongoing commuication with interns. The program included its classical components: *Summer Seminar Series* and *Summer Proceedings*, along with new ones: *Virtual Poster Blitz*, *Reading Groups*, and *Diversity & Inclusion Workshop*. Below are more details about each activity.

2. Seminar Series. An important educational component of the summer program is the CSRI Summer Seminar Series. The Seminar Series' focus areas for 2020 talks were: *Artificial Intelligence* (AI) and *Neuromorphic Computing* (NC)—which are well aligned with Sandia's strategic thrusts in computer and information sciences. Expert staff members from 1400—CCR, 5900—Proliferation Assessment, 6300—Systems Mission Engineering, 6600—Asset Security & WMD Response, and 8700—Homeland Sec. & Def. Systems have shared their knowledge and expertise on a variety of AI/MC algorithms/methods/applications with our interns, using different presentation and video technologies as shown in Figure 2.1. We would like to thank the staff who spoke at the 2020 Seminar Series, listed in Table 2.1. We would also like to thank Suma Cardwell and Craig Vineyard for coordinating the 1421-led lightning talks on neuromorphic computing methods and applications.

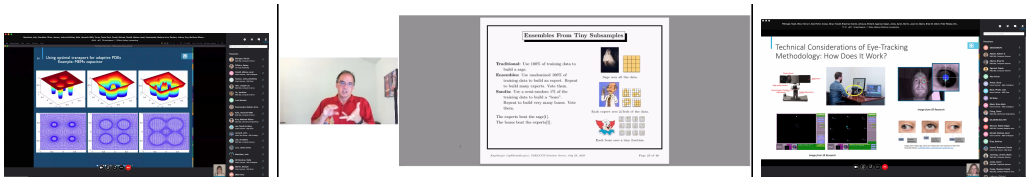


Fig. 2.1: Skype screenshots from different seminar series' talks.

Table 2.1: List of talks and speakers at the 2020 Seminar Series.

Date	Name	Org	Title
6/9	Team	1400	Introduction to the CSRI Summer Program 2020
6/16	Brad Aimone	1421	Overview of Neuromorphic Computing Research
	Frances Chance	1421	Dragonfly Inspired Interception
	Suma Cardwell	1421	Neuromorphic Hardware and Architectures
6/23	Craig Vineyard	1421	Introduction to AI/ML efforts and neural approaches
	William Severa	1421	Computing Challenges in Connected and Autonomous Vehicles
	Felix Wang	1421	Brain-Inspired Navigation
6/30	Erin Acquesta	5954	Epidemiology Modeling of the COVID-19 Pandemic
7/14	Thomas Catanach	8759	A Bayesian Perspective on Machine Learning and UQ
7/16	David Stracuzzi	1462	The Role of Uncertainty Quantification in Machine Learning
7/21	Danny Dunlavy	1461	Tensor Decompositions for Analyzing Multi-Way Data
7/30	Philip Kegelmeyer	8700	The Counter-Intuitive Properties of Ensembles for Machine Learning: Democracy Defeats Meritocracy
8/4	Mallory Stites	6672	Using Eye-Tracking to Understand Cognitive Processing
8/6	Kelsey DiPietro	1463	Optimal Transport and its applications in adaptive transport and machine learning
8/13	Marta D'Elia	8754	Data-Driven Physics-Informed Approaches for Model Learning in the Context of Nonlocal Equations
8/18	Tian Ma	6300	Introduction to Remote Sensing Object Detection
8/20	Tim Draelos	6362	Deep Learning – Engineered, Data Driven AI: Applications and Opportunities
8/25	Warren Davis	1461	Deep Learning Applications

3. Proceedings. All students and their mentors were encouraged to contribute a technical article to the CSRI Proceedings. In many cases, the CSRI Proceedings are the first opportunity that students have to write a research article. Not only do these proceedings serve to document the research conducted during the summer but, as part of the research training goals of Sandia, it is the intent that these articles serve as precursors to or first drafts of articles that could be submitted to peer-reviewed journals. As such, each article has been reviewed by a Sandia staff member knowledgeable in that technical area with feedback provided to the authors.

Contributions to the 2020 CSRI Proceedings came from different centers and have been organized into the following broad categories— *Computational & Applied Mathematics*, *Software & High Performance Computing* and *Applications*; illustrated in Figure 3.1.

We would like to thank all participants who have contributed to the outstanding technical accomplishments of the proceedings in 2020. We would also like to thank those who reviewed articles for this proceedings; their feedback is an important part of the training process and has significantly improved the proceedings quality: Brad Aimone • Park Hays • John Jakeman • Heidi Thornquist • Siva Rajamanickam • Brian Kelley • Eric Phipps • Andrew Bradley • Kathryn Maupin • Kyungjoo Kim • Cannada Lewis • Matthew Wong • Edward Walsh • Felix Wang • Suma Cardwell • Patrick Widener • Erik Boman • Chuck Hemberee • Marta D'Elia • Andrew Baczewski • Paul Kuberry • Nat Trask • Chad Sockwell • Oksana Guba • Mohan Sarovar • Chris LaFleur • Brian Ehrhart • Joey Hart • Justin Winokur • Brian Adams • Scott Hemmert • Kara Peterson • Richard Barrett • Jon Berry • Rich Lehoucq • Christian Glusa • Jonathan Hu.

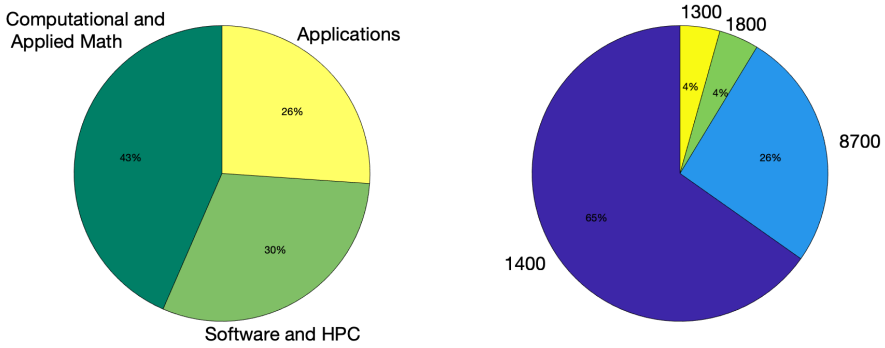


Fig. 3.1: Articles at CSRI Proceedings 2020, by research area and by intern organization.

4. Virtual Poster Blitz. In preparation for the proceedings, a *virtual poster blitz* event was held on 7/23/2020, where all CSRI interns (hired under the CSRI postdoc posting) were invited to participate by submitting a summary slide as illustrated in Figure 4.1. Other interns, including interns with other programs, were also invited to submit a slide with their mentor's approval. Interns used this event as an opportunity to formalize their work direction(s), socialize their ideas, and network with other interns and staff across the labs. The slides from the virtual poster blitz event are published as as SAND report (SAND2020-7414 PE).

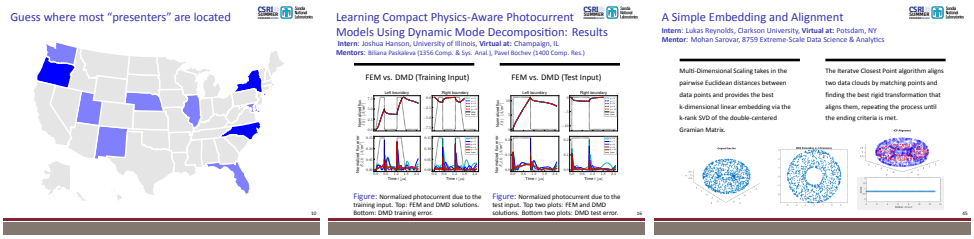


Fig. 4.1: Example slides from the Virtual Poster Blitz event.

5. Reading Groups. Reading Groups offered students an optional opportunity to deep dive into recent technical papers in the fields or Artificial Intelligence and Machine Learning (AI/ML). We would like to thank Siva Rajamanickam (1465) and Eric Cyr (1442) for organizing and leading the discussions. A list of papers discussed this year are:

- **Deep Learning**: LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. nature, 521(7553), 436-444.
- **LSTM**: Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780.
- **Generative Adversarial Nets**: Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., & Bengio, Y. (2014). Generative adversarial nets. In Advances in neural information processing systems (pp. 2672-2680).
- **Resnet**: He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and

pattern recognition (pp. 770-778).

- **Reinforcement learning** Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., & Dieleman, S. (2016). Mastering the game of Go with deep neural networks and tree search. nature, 529(7587), 484.
- **Batch Normalization**: Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167.
- **Graph Neural Networks**: Kipf, T. N., & Welling, M. (2016). Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907.

6. D&I Workshop. In collaboration with the Sustailable Horizons Institute (HSI), we organized a 3-hour workshop on 7/7/2020, entitled "Pathways to Success: Navigating School & Your Career". Led by Dr. Mary Ann Leung, this workshop was an online activity designed to help bridge the virtual gap in the summer experience and facilitate interaction between interns and mentors. The activity began with a game designed to help participants get to know each other, followed by an exploration of pathways to a successful career in Computational Science and Engineering. The workshop included interactive discussion and visualization of pathways from the past through the present and into the future. Participants developed insights and concrete approaches for navigating pathways to career success. They also identified ways to recognize obstacles and resources to overcome them.

Under the unusual virtual work circumstances, the success of the program and its different activities hinged on the hard work of enthusiastic students and their dedicated Sandia technical staff mentors. We, therefore, would like to thank all students and mentors for their dedication. We would also like to thank Edgar Galvan and Jerry Mcneish (8754) who coordinated the program activities in California. Furthermore, the CSRI summer program would not be possible without the administrative support of Celia Montoya, Sandra Portlock, Cookie Santamaria, Becky March, as well as Marc Campanozzi from the Skype Large Meeting team.

Ahmad A. Rushdi
Michael L. Parks

November 1, 2020

Table of Contents

III. Applications

Articles

I. Computational & Applied Mathematics

Computational & Applied Mathematics are concerned with the design, analysis, and implementation of algorithms to solve mathematical, scientific, or engineering problems. Articles in this section describe methods to design new nerual network architectures, discretize and solve partial differential equations, couple multiphysics systems of equations, and analyze sensitivity & quantify uncertainty in complex systems.

1. *Anwar, Vineyard, Severa, Musuvathy, and Cardwell* extend **Weight Agnostic Neural Networks** (WANN) to search for for spiking circuits, and in doing so investigate whether spiking circuit motifs can also yield task performance that is weight agnostic.
2. *Bergstrom, Butler, and Wildey* demonstrate that a **weighted empirical distribution function solving a constrained quadratic optimization** problem produces an approximate observation-consistent solution. They extend the formulation and solution of the optimization problem to more general cases where parameter samples are not required to be either iid or from a specified proposal distribution.
3. *Fox, Rajamanickam, and Song* present a novel **multi-resolution convolutional architecture for learning over concentric 3D spherical feature maps**, of which the single sphere representation is a special case. They highlight the applicability of their method for different types of 3D inputs.
4. *Gilbert, Rajamanickam, Madduri, and Acer* study **graph coarsening methods** for high-performance graph partitioners and PDE solvers. They evaluate the performance of four coarsening methods using: time to generate a full hierarchy of coarse graphs and bipartitioning cutsize resulting from spectral and FM refinement methods.
5. *Jones and Bosler* examine the use of **Radial Basis Function (RBF) methods for interpolation** and construction of surface differential operators for the unit sphere. They present convergence results for the approximation of the Laplace-Beltrami operator and interpolation of a spherical harmonic.
6. *Merritt, Geraci, Eldred, and Portone* propose a **hybrid MLMC-PCE approach for Global Sensitivity Analysis** (GSA). They use an MLMC sampling strategy to compute the PCE coefficients, extending the applicability of the PCE-based GSA analysis to expensive high-dimensional problems.
7. *Morales and DiPietro* introduce a **finite element form of the Monge-Ampère equation**, specifically in nonvariational form, using Intrelab, a subdirectory of the Trilinos package Intrepid. They present convergence studies of using the nonvariational finite element method for elliptic problems that cannot be put into the standard variational form.
8. *Myers, Dunlavy, Teranishi, and Hollman* examine the **sensitivity of the SparTen high performance sparse Tensor Decomposition software**. They experiment with sensitivity and computational issues on three benchmark tensor data sets, on several different CPU architectures, in order to establish generalized profiles of algorithm convergence behavior.

9. *Olson, Gulian, and D'Elia* investigate the relationship between **tempered and truncated fractional operators and the unified nonlocal diffusion operator** and examine computationally cheap alternatives to tempered fractional operators.
10. *Voronin, Tuminaro, Olson, and MacLachlan* develop practical and theoretical guidance into different **multigrid preconditioners for systems of partial differential equations**. They experiment with several Algebraic MultiGrid (AMG) variants, showing significantly different convergence histories.

A.A. Rushdi
M.L. Parks

November 1, 2020

EVOLVING SPIKING CIRCUIT MOTIFS USING WEIGHT AGNOSTIC NEURAL NETWORKS

ABRAR ANWAR*, CRAIG M. VINEYARD†, WILLIAM M. SEVERA‡, SRIDEEP MUSUVATHY§, AND SUMA CARDWELL¶

Abstract. Neural networks have increasingly been applied as state-of-the-art solutions to tasks ranging from image and video analysis, to natural language processing, to strategic planning and control. These investigations have yielded many different neural network architectures as various optimizations are pursued with the objectives of improved performance as well as to improve computational costs. Furthering this exploration, neural architecture search (NAS) has emerged as an algorithmic method of developing neural network architectures. Weight Agnostic Neural Network (WANN) is an evolutionary-based NAS approach. Fundamentally, WANN pursues circuit motifs which enable decent performance on tasks largely due to the network structures that are relatively insensitive to weights and typically much smaller than an equivalent performance dense network. Here we extend the WANN framework to search for spiking circuits, and in doing so investigate whether spiking circuit motifs can also yield task performance that is weight agnostic. In doing so, we analyze properties such as the the complexity of the solution and performance. Our results successfully show the performance of spiking WANNs on several exemplar tasks.

1. Introduction. Neural networks are becoming exceedingly commonplace; however, limitations of traditional hardware which neural networks run on are becoming apparent, specifically in the low-power domain. For edge computing applications, such as drones, satellites, and micro-robots, running larger neural networks is not feasible due to the energy cost. Neuromorphic computing introduces a new paradigm for computing that is brain inspired with an added benefit of low energy usage.

In many cases, neural networks tend to be overparameterized. Recently, a shift towards pruning deep neural networks to make them sparser has become common. In addition, NAS has also been effective in finding architectures that reduce complexity and increase performance of neural networks [7, 22, 13, 5]. For spiking neural networks, Evolutionary Optimization for Neuromorphic Systems (EONS) [17] is such an approach to generate spiking neural networks. Recent work in searching for sparse topologies for various tasks in classical neural networks showed that neural network weight training can be skipped, as a universal parameter sharing approach is effective in evaluating the success of a potential network topology. We use this approach to find topologies in spiking neural networks for solving MNIST, swingup cartpole, bipedal walker, and Atari Atlantis problems. This work provides evidence that spiking networks benefit from weight agnostic graph structures in the same way scalar-weight networks do.

In Section 2, we provide a short background on neuromorphic computing, spiking networks, neural architecture methods, and Weight Agnostic Neural Networks. In Section 3, we define the spiking WANN, followed by the results on various tasks in Section 4. Lastly, Section 5 discusses considerations for future applications and work on spiking WANNs.

2. Background.

2.1. Neuromorphic Computing.

Neuromorphic computing relies on event-based spiking communication between neurons. Conversely, a typical artificial neural network (ANN) relies on dense communication of continuous values. In order for ANNs to work with this new paradigm, they must be converted into spiking neural networks (SNNs). The

*University of Texas at Austin, abraranwar@utexas.edu
†Sandia National Laboratories, cmviney@sandia.gov
‡Sandia National Laboratories, wmsever@sandia.gov
§Sandia National Laboratories, smusuva@sandia.gov
¶Sandia National Laboratories, sgcardw@sandia.gov

main motivation motivation for this difference is the promise of energy-efficient compute evidenced by biological systems. Hence SNNs try to replicate this by communicating in a fashion loosely inspired by biological neurons. We can define a spiking neuron computation by a threshold activation function. Although a binary threshold ANN is not strictly an SNN as it does not include the temporal domain, since it is compatible with neuromorphic hardware, it is referred to as such. Severa et al. [18] noted that converting ANNs to SNNs for neuromorphic computing is a non-trivial process, thus they iteratively sharpen various activation functions to be binary. We show that evolved weight agnostic neural networks with binary activation functions perform well and should be suitable to be transferred onto neuromorphic hardware.

2.2. Lottery Ticket Hypothesis and Network Pruning. Network pruning focuses on removing connections to create sparse networks that have a smaller number of connections and weights. Pruning typically requires prior training, and then reducing the number of weights [2]. The lottery ticket hypothesis solves the difficult problem of training sparse networks. It states that a randomly initialized neural network has a sparse subnetwork that performs just as well, if not better than its dense counterpart [8]. Building on this finding, it was discovered that these pruned networks perform better than chance with randomly initialized weights [21], further supporting the idea that the network topology influences performance.

2.3. Neural Architecture Search. In contrast to pruning methods, the goal of neural architecture search (NAS) is to learn a network topology that can achieve good performance on certain tasks, while sometimes ensuring a lower number of parameters. Zoph and Le's [22] pioneering work in NAS showed the intense computational resources needed to generate accurate neural networks due to the large search spaces. Most NAS approaches are split into three separate components: the search space, the search algorithm, and the evaluation strategy [7]. The search space consists of a set of operations such as convolutions or pooling layers and how these operators can be appended to form network topologies. The search algorithm is how NAS methods selects candidates from a population of network architectures and how they optimize these candidates. The evaluation strategy is where the performance of the models are evaluated, either by actually running the network or using some other metrics to estimate performance.

Typically, during the evaluation stage, if the algorithm needs to evaluate a test set, the network to first undergo training. This makes the evaluation strategy the most expensive part of the operation. Parameter sharing is one approach used to gain a speed up [15], where child models share parameters with their parent. Brock et al. [3] uses a HyperNet [10] to generate weights based on the encoding of the network architecture parameters. Weight agnostic neural networks (WANNs) [9] take an approach inspired by evolutionary methods to evolve neural network topologies, focusing on individual nodes rather than a set of operations.

2.4. Weight Agnostic Neural Networks. Weight agnostic neural networks are inspired by the fact precocial species can accomplish several tasks at birth, such as duck hatchlings being able to swim and eat [20]. WANNs follow an iterative topology search algorithm inspired by the NEAT evolutionary search method [19]. In most architecture search approaches, each generated topology requires individual weight training, which tends to be the most expensive portion of the algorithm. WANNs show that the network topology is important by enforcing weight-sharing across the whole network. Rather evaluating a network by its performance on a test set after training, WANNs evaluate on a set of shared weight values.

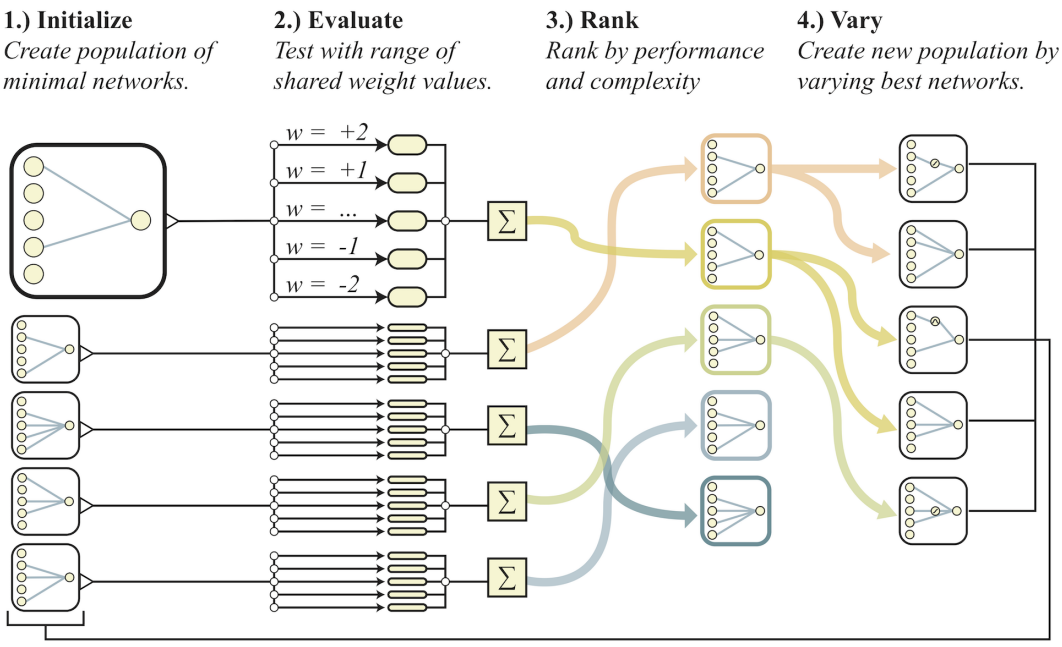


Fig. 2.1: Visualization of topology search used to evolve WANNs. Sourced from WANN paper [9]

To detail the approach in Figure 2.1, the algorithm goes as follows:

1. A population of various network topologies are generated.
2. For reinforcement learning/control tasks, the network runs through several rollouts, each using a different shared weight value. For classification, it simply evaluates the training set using the various shared weight values.
3. The networks are ranked in regards to their average performance and the number of connections as a loose estimate of model complexity.
4. The top networks reproduce by adding or mutating connections and activation functions.

The ranking process uses the crowding distance metric from Non-Dominated Sorting Genetic Algorithm II (NSGA-II) [6], an evolutionary approach to multi-objective optimization. Two objectives are optimized over: the mean fitness across each of the iterations and an alternating objective of max fitness and the number of connections. The number of connections is minimized 80% of the time while the max fitness is maximized 20% of the time. This is to ensure that the network is able to grow in complexity if it leads to increased performance. The best performing network is chosen as the final network; however, there does exist a Pareto frontier of individual networks between network complexity and performance.

The mutation process involves adding new connections with random activation functions, changing existing activation functions, or adding a connection between two existing activation functions. The set of available activation functions are linear, step (binary/threshold), sin, cosine, Gaussian, tanh, sigmoid, inverse, absolute value, and ReLU. Though Gaier and Ha admit that they did not experiment much on the number of activation functions, they speculated that the variety of activation functions allowed for decent performance from the WANNs; however, as we will see, simply two activation functions are effective.

3. Spiking WANNs. The overall approach to generating Spiking WANNs is the same as the search in Figure 2.1. The evaluation step of the search is highly paralellizable and is asynchronously evaluated across hundreds of processes. A reduced set of activation functions are used, namely the threshold and linear activation functions. Threshold activation functions themselves can easily be transferred onto neuromorphic hardware; however when combined with a linear activation function, they mimic additive dendritic trees and can be approximated by leaky integrate-and-fire neurons with delays. We recognize that the inputs and outputs of our network may not be fully spiking; however, this can be overcome using approximating networks and/or expanding codings.

Fig. 4.1: The three tasks run are the swingup cartpole task [4] (left), the bipedal walker task [4] (center), and MNIST classification [12] (right)

4. Expiremental Results.

4.1. Tasks. We evaluated primarily four tasks. The first was a cartpole swingup task [4]. The cartpole task is a classic continuous control problem where a pole starting in an upright position must be balanced. The swingup version of this task starts in a resting position with the pole hanging down and needs to be swung upright and balanced, and unlike its simpler counterpart, cannot be solved using a linear controller. The input is angle of the pole, sines/cosines of the angle, and the x coordinate. The expected output is the force of ± 1.

The second task was the BipedalWalker-v2 task for OpenAI Gym [4]. The goal of the task is have a bipedal agent navigate across randomly generated terrain. A positive reward is awarded for distance, while a negative reward for motor torque is given to ensure efficient motions are made. The input is the state of the agent, consisting of the various speeds and positions of different joints and ten LiDAR measurements. Overall, the input consists of 24 dimensions.

The third task was MNIST digit classification [12]. Although for most computer vision tasks, MNIST is low in dimensionality, due to evolutionary approaches requiring making connections at random, convergence can take a long time. Standard MNIST is 28x28, which was reduced to 16x16 to reduce the dimensions.

The fourth task is the Atari Atlantis task, one of the many well-known games in the reinforcement learning community. These games became a prominent RL benchmark starting in 2013 when Mnih et al. [14] published their seminal work on DQN approaches surpassing human performance.

4.2. Comparison to WANNs. Experiments were run on each task using the same parameters from the WANN paper, as seen in Table 4.1, to ensure fair comparisons between them.

The results in Table 4.2 use the reward metric averaged over 100 rollouts for the relevant control tasks along with their standard deviations for the best evolved network topology.

Table 4.1: Parameters used for each task

Task	# of Generations	Population Size
Swingup Cartpole	1024	192
Bipedal Walker	2048	480
MNIST	4096	960

Table 4.2: Results for the various tasks.

	WANN		
	Tuned Shared Weight	Tuned Weights	# of Connections
Swingup Cartpole	723 ± 16	932 ± 6	62
Bipedal Walker	261 ± 58	322 ± 7	338
MNIST	91.9%	94.2%	1228
	Spiking WANN		
	Tuned Shared Weight	Tuned Weights	# of Connections
Swingup Cartpole	745 ± 11	912 ± 5	56
Bipedal Walker	290 ± 22	281 ± 31	210
MNIST	87.7%	88.2%	576

For MNIST classification, the accuracy is given on the test set. The tuned shared weight category is the best shared weight value for the evolved network topology. The tuned weights is when the network's weights are individually trained using a population-based REINFORCE algorithm. The tuned shared weights results are comparable to the original WANN, but there is a degradation in performance when converting into the spiking-like WANN. Other ANN to SNN conversion methods have also shown performance degradation during the switch to threshold activation functions [18].

Interestingly, the tuned shared weights for the spiking WANNs have generally higher performance than the WANN, but the finetuned weights perform worse. This can potentially be attributed to fewer number of weights to finetune, as we see spiking WANNs consistently generate smaller networks.

4.3. Classification. With good results on reinforcement learning tasks, Gaier and Ha explored the capability of WANNs in MNIST classification. They state that classification is unforgiving, as the algorithm is either right or wrong; there is no possibility of recovery as there is in an episode in RL tasks. Table 4.2 shows the performance to be worse across the board for the classification task. Again, this might be related to the significantly smaller sized network generated by the spiking WANN.

4.4. Multi-Objective Optimization. Although all results in Table 4.2 show the best individual, there exists a set of Pareto-optimal solutions since it's a multi-objective optimization problem. Figure 4.2 shows all individuals over the evolutionary process. We can see a Pareto frontier develop on the right hand side of the graph, as we are minimizing the number of connections and maximizing the mean fitness. As generations increase, we see the number of connections are increasing, which is easily noticable by the gradient towards red. The charts only plot the mean fitness and the number of connections; however, the algorithm does an alternating objective optimization, where 20% of the time, the number of connections objective is swapped out with a maximization of peak fitness. This is to encourage growth in the number of connections, as well as performance.

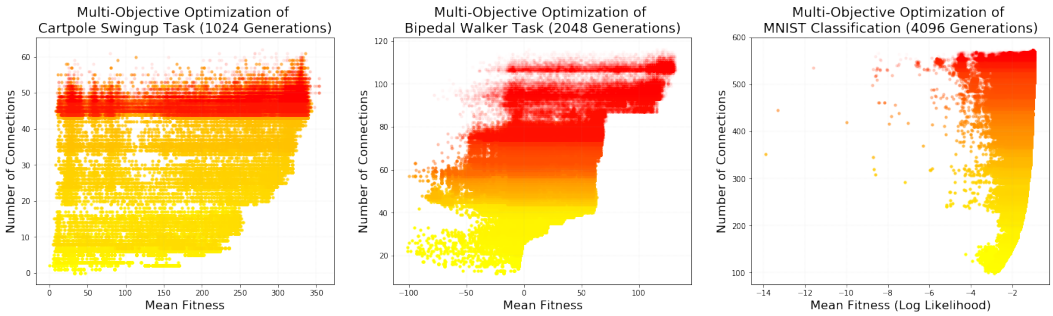


Fig. 4.2: Fitnesses of individuals across multiple generations. The color map is from yellow to red, where the more red a point is, the later generation it comes from

Table 4.3: Results for Atari Atlantis task. Average Human and Random Agents results sourced from [1]. DQN and HyperNEAT results sourced from [16].

Game	Spiking WANN	Average Human	Random Agent	DQN	HyperNEAT
Atlantis	51180.0	29028.1	12850.0	76108.0	61260.0

The color gradient for the cartpole swingup task looks odd due to the lack of a gradient. This is because the cartpole task reached its objective significantly earlier, as seen in Figure 4.3. We see a clear correlation between the number of connections and the fitness values. This is further justification on why the agent is encouraged to ignore the number of connections a certain percentage of the time.

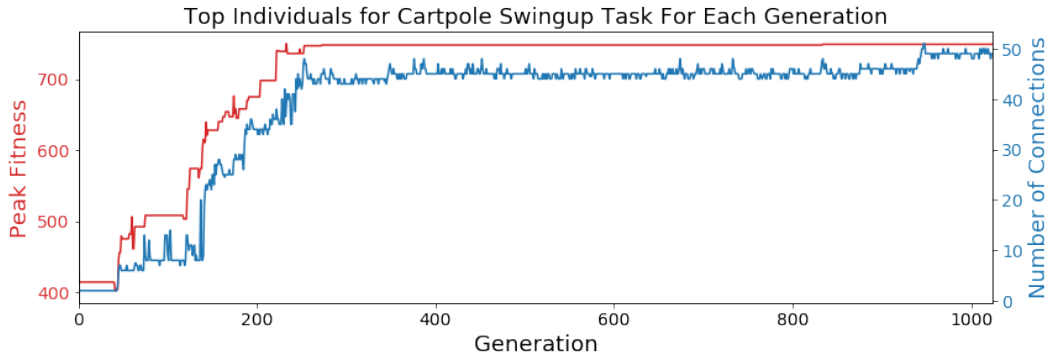


Fig. 4.3: Peak fitness and number of connections for the best individual in each generation. The red peak fitness lines' score is on the left while the number of connections is blue and on the right hand side.

4.5. Atari. Testing on the Atari game, Atlantis, we were able to see that, even without considerable extensions, WANNs are capable of achieving DQN-like performance at the fraction of the computational cost. Due to the high dimensionality of the frames, the input was fed through a ResNet trained on ImageNet, whose final layer was cut off and fed in as

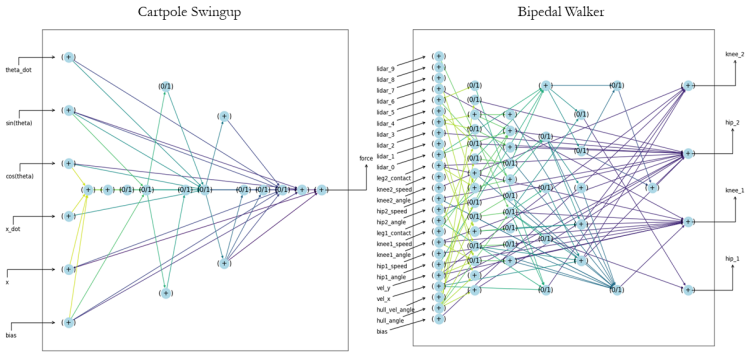


Fig. 4.4: Network topologies of the top individuals for the cartpole swingup task (left) and the bipedal walker task (right).

the input to the WANN. This method would allow the spiking WANN to converge faster from the smaller input space.

The results, as seen in Table 4.3, shows the reward given for the task across various agents. The longer the agent is able to play, the performance increases. The Spiking WANN results are comparable to a DQN, as well as HyperNEAT from Hauskenect et al. [11]. HyperNEAT is a neuroevolution method which evolves an artificial neural network topology using compositional pattern producing networks, allowing for it to efficiently handle large input sizes. The spiking WANN score shows a slight loss in performance compared to the other two methods, but clearly beats a random agent and the average human. The spiking WANN network for this task is using a shared fixed weight rather than finetuned weights due to time constraints. A slight performance boost should occur if the weights are individually trained, as seen in the previous tasks. In addition, the performance of the network is after only 64 generations, where increased generations are likely to increase the performance.

5. Conclusion. We hope to map these, or similar networks, to physical neuromorphic hardware. Again, some inputs and outputs may not be fully spiking, such as the softmax operation used for classification tasks. Methods around this will be useful to explore. In addition, the complexity metric used was the number of connections. This is meant to be a loose approximation of energy usage, but different target architectures perform differently with different network topologies. Exploring energy-based constraints by changing the complexity metric to be true to the target platform would allow for neural network-hardware co-design. In addition, exploring the use of WANNs on a broader set of classification tasks and reinforcement learning tasks would allow us to evaluate its capabilities. It may be worthwhile to investigate changing the parameters, as these spiking-like networks have far fewer activation functions available.

Whetstone refines the activation functions of a typical deep neural network to become threshold activation functions, which can then be used on neuromorphic hardware. Leveraging the representational capabilities of a Whetstone network with a spiking WANN may increase performance on datasets with large input sizes.

There also exists a potential for noise resilience. The evolutionary process for developing the network topology ideally generates networks robust to noise, potentially in the input

space or in the synaptic weights. Future exploration of this domain would make it an ideal candidate for generating networks on neuromorphic hardware where the weights are noisy. Spiking WANNs have been shown to perform well on a variety of tasks. Once a spiking WANN has been implemented onto neuromorphic hardware, we hope to observe significant power savings and reduced energy consumption compared to its traditional counterparts.

6. Acknowledgment. This work was supported by DOE NA-22 funding at Sandia National Laboratories.

REFERENCES

[1] A. P. Badia, B. Piot, S. Kapturowski, P. Sprechmann, A. Vitvitskyi, D. Guo, and C. Blundell, *Agent57: Outperforming the atari human benchmark*, 2020.
[2] D. Blalock, J. J. Gonzalez Ortiz, J. Frankle, and J. Guttag, *What is the state of neural network pruning?*, in Proceedings of Machine Learning and Systems 2020, 2020, pp. 129–146.
[3] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, *SMASH: one-shot model architecture search through hypernetworks*, CoRR, abs/1708.05344 (2017).
[4] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, *Openai gym*, CoRR, abs/1606.01540 (2016).
[5] H. Cai, L. Zhu, and S. Han, *Proxylessnas: Direct neural architecture search on target task and hardware*, CoRR, abs/1812.00332 (2018).
[6] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, *A fast and elitist multiobjective genetic algorithm: Nsga-ii*, IEEE transactions on evolutionary computation, 6 (2002), pp. 182–197.
[7] T. Elsken, J. H. Metzen, and F. Hutter, *Neural architecture search: A survey*, 2018.
[8] J. Frankle and M. Carbin, *The lottery ticket hypothesis: Training pruned neural networks*, CoRR, abs/1803.03635 (2018).
[9] A. Gaier and D. Ha, *Weight agnostic neural networks*, (2019).
[10] D. Ha, A. M. Dai, and Q. V. Le, *Hypernetworks*, CoRR, abs/1609.09106 (2016).
[11] M. Hausknecht, J. Lehman, R. Miikkulainen, and P. Stone, *A neuroevolution approach to general atari game playing*, IEEE Transactions on Computational Intelligence and AI in Games, 6 (2014), pp. 355–366.
[12] Y. LeCun and C. Cortes, *MNIST handwritten digit database*, (2010).
[13] H. Liu, K. Simonyan, and Y. Yang, *DARTS: differentiable architecture search*, CoRR, abs/1806.09055 (2018).
[14] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, *Playing atari with deep reinforcement learning*, CoRR, abs/1312.5602 (2013).
[15] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, *Efficient neural architecture search via parameter sharing*, 2018.
[16] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, *Evolution strategies as a scalable alternative to reinforcement learning*, 2017.
[17] C. D. Schuman, J. P. Mitchell, R. M. Patton, T. E. Potok, and J. S. Plank, *Evolutionary optimization for neuromorphic systems*, in Proceedings of the Neuro-Inspired Computational Elements Workshop, NICE '20, New York, NY, USA, 2020, Association for Computing Machinery.
[18] W. Severa, C. M. Vineyard, R. Dellana, S. J. Verzi, and J. B. Aimone, *Training deep neural networks for binary communication with the whetstone method*, Nature Machine Intelligence, 1 (2019), pp. 86–94.
[19] K. O. Stanley and R. Miikkulainen, *Evolving neural networks through augmenting topologies*, Evolutionary Computation, 10 (2002), pp. 99–127.
[20] J. M. Starck and R. E. Ricklefs, *Patterns of development: the altricial-precocial spectrum*, Oxford Ornithology Series, 8 (1998), pp. 3–30.
[21] H. Zhou, J. Lan, R. Liu, and J. Yosinski, *Deconstructing lottery tickets: Zeros, signs, and the supermask*, in Advances in Neural Information Processing Systems 32, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds., Curran Associates, Inc., 2019, pp. 3597–3607.
[22] B. Zoph and Q. V. Le, *Neural architecture search with reinforcement learning*, CoRR, abs/1611.01578 (2016).

ESTIMATING OBSERVATION-CONSISTENT SOLUTIONS USING WEIGHTED EMPIRICAL DISTRIBUTION FUNCTIONS

KIRANA O. BERGSTROM*, TROY D. BUTLER†, AND TIM M. WILDEY‡

Abstract. The class of stochastic inverse problems considered in this work involves the characterization of a probability measure on the input parameters of a computational model whose subsequent push-forward matches an observed probability measure on specified quantities of interest associated with model outputs. Such a solution is formally defined by the pullback of the observed probability measure and is therefore referred to as an observation-consistent solution. A probability measure may be quantitatively characterized and approximated in several ways. Previous approaches for approximating observation-consistent solutions relied upon density estimation or set/event approximations. Such approaches are challenging to implement in scenarios where the number of either simulated or observational data are limited. Separate research has tackled the problem of estimating push-forward measures under such scenarios by using weighted empirical distribution functions where the weights are defined as the solution to a constrained quadratic optimization problem. This leads to the major contributions of this work. We demonstrate that a weighted empirical distribution function solving a constrained quadratic optimization problem produces an approximate observation-consistent solution. Additionally, we extend the formulation and solution of the optimization problem to more general cases where parameter samples are not required to be either independent identically distributed or from a specified proposal distribution.

1. Introduction. Inverse problems seek to learn information about the parameters of a model from observed data. The precise nature of the inverse problem depends on the appropriate characterization of the model, data, uncertainties, etc. In some cases, these problems are posed deterministically and solved using optimization-based methods. In other cases, an epistemic characterization of likely parameter values is sought through a Bayesian formulation. In this paper, we consider a different class of problems where we seek to determine probabilistic information about the parameters of the model from probabilistic information of the model outputs. In other words, we seek an aleatoric characterization of the model inputs given an aleatoric characterization of the observed data. The solution to this inverse problem is required to be *observation-consistent*, meaning that the solution to the inverse problem, defined as a probability distribution on the parameter space, propagates through the model to a given target distribution on observations. This type of inverse problem naturally arises in scenarios where the variability in the observations is primarily due to intrinsic variability in the model inputs rather than measurement noise or error.

Previous approaches to solving this problem either seek to approximate a pullback measure directly through event approximation in both the input and output spaces [5] or to use accept-reject sampling based on a (non-parametric) density approximation in the output space [6]. These approaches are challenging to implement when the number of simulated data are limited, e.g., when the computational model is expensive to evaluate. Moreover, there is an implicit assumption that sufficient information exists on observations to specify either a measure or density (or possibly a family of observed measures or densities).

In this work, we demonstrate that an optimization-based, distribution-matching method can be used to approximately characterize pullback measures even in scenarios where simulated or observational data are limited. The method is based on the approach developed in [1] for approximating push-forward measures by performing a change-of-measure objective using empirical distribution functions (EDFs) and solving an optimization problem to determine optimal weights on the simulated output samples. These weights are designed such that when they are used to form a weighted EDF, they minimize the L_2-norm be-

*Sandia National Labs and University of Colorado Denver, kirana.bergstrom@ucdenver.edu
†University of Colorado Denver, troy.butler@ucdenver.edu
‡Sandia National Labs, Center for Computing Research, tmwilde@sandia.gov

tween the weighted EDF and a given target distribution function. Such optimal weights are guaranteed to exist using the L_2-norm as this results in a quadratic optimization problem. Thus, for the case where simulated data are limited, a solution is achievable that is optimal in an L_2-sense. Moreover, the target distribution may be in the form of either a cumulative distribution function (CDF) or an EDF. Thus, for the case where observational data are limited, the approach produces an L_2-optimal solution associated with the observed EDF.

The optimization-based method developed in [1] assumes that the parameter samples in the parameter/output sample pairs are random independent identically distributed (i.i.d.) samples generated from a proposal distribution on the parameter space. This assumption may not be satisfied when the input/output sample sets are generated offline, possibly in an unknown or semi-structured manner (e.g., a parameter sweep study or a grid-based study). Another contribution of this work is the development of a modified version of the L_2-optimal weighting method that can handle arbitrary sets of input/output samples while incorporating proposal distribution information. This involves a two-stage optimization-based procedure where L_2-optimal weights are first found on the input data and then a second optimization problem is used to determine optimal weights on the output data. This second optimization problem includes an additional constraint to maintain the structure inherited from the first optimization.

The remainder of this paper is organized as follows. Section 2 introduces the model and the class of inverse problems studied in this paper. Section 3 reviews the L_2-optimal empirical distribution approach developed in [1] and introduces the utilization of this method to approximately characterize a pullback probability measure. In Section 4, we discuss the extension of the L_2-optimal empirical distribution approach to the more general case with arbitrary input samples. Applications of the framework are given in Section 5 and our concluding remarks are in Section 6.

2. A stochastic inverse problem and observation-consistent solutions. Let $(\Lambda, \mathcal{B}_\Lambda)$ be a measurable space of input parameters of interest for a model, where $\Lambda \subset \mathbb{R}^p$ and $\mathcal{B}_\Lambda$ is the Borel σ-algebra of Λ. We consider a function Q that maps parameters $\lambda \in \Lambda$ to the quantities of interest (QoI) associated with observable model outputs. Let $\mathcal{D} = Q(\Lambda) \subset \mathbb{R}^d$ denote the data space. Assume that $Q : \Lambda \to \mathcal{D}$ is a measurable mapping from the parameter space to the output space $(\mathcal{D}, \mathcal{B}_\mathcal{D})$, where $\mathcal{B}_\mathcal{D}$ is the Borel σ-algebra of $\mathcal{D}$.

Given a *target* probability measure, $\mathbb{P}_{targ}$, on $(\mathcal{D}, \mathcal{B}_\mathcal{D})$, the stochastic inverse problem seeks to construct a pullback probability measure, $\mathbb{P}_\Lambda$, on $(\Lambda, \mathcal{B}_\Lambda)$. Such a measure is referred to as an *observation-consistent* solution since it has the property that

$$\mathbb{P}_\Lambda(Q^{-1}(A)) = \mathbb{P}_{targ}(A), \ \forall A \in \mathcal{B}_\mathcal{D}. \tag{2.1}$$

The disintegration theorem [8] provides insight into the existence, uniqueness, and structure of a pullback measure. For the purposes of this work, it suffices to say that the disintegration of a pullback measure is defined in terms of a marginal probability measure that is uniquely determined by $\mathbb{P}_{targ}$ and a family of conditional probability measures that must be specified on the generalized contours given by $Q^{-1}(q)$ for a.e. $q \in \mathcal{D}$.

In [5], an ansatz is used to specify the conditional probability measures and probabilities on the approximate partitioning of contour events are used to estimate a pullback measure. When the parameter space is high-dimensional or the contour events have complex geometric structures, naive implementations of this approach can easily require an intractable number of evaluations of the map. In [6], the push-forward of an initial probability density defined on $(\Lambda, \mathcal{B}_\Lambda)$ is used to specify conditional probability densities and represent a pullback measure in terms of an update to the initial density. While the approximation and evaluation of

densities occur in $\mathcal{D}$, which is often much lower-dimensional than Λ, this may still require a large number of model evaluations to produce accurate density estimates. Moreover, in both approaches it is assumed that the target measure/density can be specified analytically or reasonably approximated from data. Thus, there is a need for new approaches that can produce accurate, or optimal, approximate characterizations of a pullback probability measure in data-sparse scenarios.

3. L_2-Optimal Weighted Empirical Distribution Functions and Application to Pullback Measures. We begin with a set of i.i.d. samples, $\{\lambda^{(1)}, \ldots, \lambda^{(n)}\}$, from the proposal distribution on the parameter space that are propagated through the QoI map such that $Q(\lambda^{(1)}) = q^{(1)}, \ldots, Q(\lambda^{(n)}) = q^{(n)}$. We may then interpret the samples, $\{q^{(i)}, \ldots, q^{(n)}\}$, as a set of n i.i.d. samples from an implicitly defined proposal distribution on the data space. Using $\preceq$ for the component-wise inequality and $\mathbb{I}(\cdot)$ as the standard indicator function, the empirical distribution function for these samples is defined as

$$F^{\mathcal{D}}_{prop;n}(q) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}(q^{(i)} \preceq q) \tag{3.1}$$

and is an unbiased estimator of the true initial CDF $F^{\mathcal{D}}_{prop}(q)$ that converges a.e. for all continuity points as $n \to \infty$. We seek weights $\{w^{(1)}, \ldots, w^{(n)}\}$ such that the weighted EDF,

$$F^{\mathcal{D}}_{prop;\boldsymbol{w}}(q) = \frac{1}{n}\sum_{i=1}^{n} w^{(i)}\mathbb{I}(q^{(i)} \preceq q), \tag{3.2}$$

defines a valid probability distribution, and is the closest such distribution in the L_2 sense to the target CDF, $F^{\mathcal{D}}_{targ}$, on the data space. This is accomplished through a straightforward application of the method from [1]. We solve the constrained minimization problem

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}\left\|F^{\mathcal{D}}_{targ}(q) - F^{\mathcal{D}}_{prop;\boldsymbol{w}}(q)\right\|_2^2 \\ \text{subject to} & \boldsymbol{w} \succeq \mathbf{0} \\ & \mathbf{1}^T\boldsymbol{w} = n, \end{array} \tag{3.3}$$

where the ith component of $\boldsymbol{w}$ is $w^{(i)}$ and the constraints enforce that $F^{\mathcal{D}}_{prop;\boldsymbol{w}}$ defines a valid distribution. In [1], the weighted proposal distribution was shown to converge in the L_1 sense to the target distribution, i.e.,

$$\lim_{n\to\infty}\int_A \left|F^{\mathcal{D}}_{targ}(q) - F^{\mathcal{D}}_{prop;\boldsymbol{w}}(q)\right| dq = 0, \tag{3.4}$$

where A is a bounded set representing the support of the target distribution.

We solve the minimization problem (3.3) using the QP solver from the Python package `cvxopt` [2]. For ease of presentation, the equations in this section utilize an exact target CDF $F^{\mathcal{D}}_{targ}$, but the analysis immediately applies to the case where an EDF, $F^{\mathcal{D}}_{targ;m}$ defined by m i.i.d. samples from $F^{\mathcal{D}}_{targ}$, is substituted for this CDF.

Computing the weights on the data space amounts to performing a change-of-measure from the push-forward of the proposal to the target distribution. By applying the weights $\boldsymbol{w}$ on the set of parameter samples, we define the weighted EDF on Λ,

$$F^{\Lambda}_{prop;\boldsymbol{w}}(\lambda) = \frac{1}{n}\sum_{i=1}^{n} w^{(i)}\mathbb{I}(\lambda^{(i)} \preceq \lambda). \tag{3.5}$$

The motivation for this work is the conjecture that as $n \to \infty$, $F^{\Lambda}_{prop;\boldsymbol{w}}$ converges to the distribution of a pullback measure with conditional probability measures on generalized contours uniquely defined by the proposal distribution on Λ.

3.1. An illustrative example. We consider the following simple problem: $\Lambda = [-1,1] \times [-1,1]$, $Q(\lambda) = \lambda_1 + \lambda_2$, and $\mathcal{D} = [-2,2]$. The proposal distribution on the parameter space is uniform $\mathcal{U}(-1,1)$ in both directions. The target distribution on the data space is $\mathcal{N}(0, 0.08)$. The result of the optimization on $\mathcal{D}$ is shown in the left-plot of Figure 3.1 for a sample size $n = 100$. We see that $F^{\mathcal{D}}_{prop;\boldsymbol{w}}$ is an excellent agreement with the target CDF. To better illustrate the resulting structure of $F^{\Lambda}_{prop;\boldsymbol{w}}$, we plot the log-scale of the weights associated with $n = 2500$ parameter samples in the right-plot of Figure 3.1.

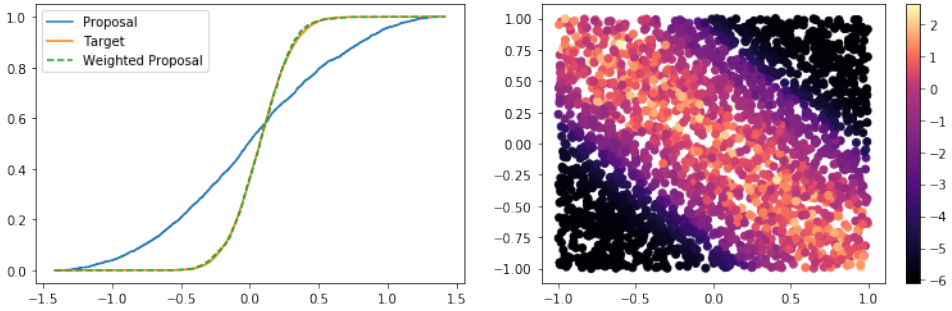


Fig. 3.1: Distributions on $\mathcal{D}$ and log of weights for $F^{\Lambda}_{prop;\boldsymbol{w}}$ on Λ ($n = 2500$).

In Figure 3.1, we see that optimization weights only need to vary in certain directions over Λ to allow the target and reweighted target densities to match in $\mathcal{D}$. We refer to these as the data-informed directions. For this example the data-informed direction is conceptualized by the line $\lambda_1 = \lambda_2$, which is orthogonal to the set-valued inverses (i.e., contours) of the QoI map defined by $\lambda_1 + \lambda_2 = q$ for each $q \in \mathcal{D}$. In this direction, we see that the structure of the weights appears to be approximately normal. This is consistent with the fact that the weights have been optimized to match a normal target distribution in the data space, and the marginal distribution in this direction in the parameter space should be the inverse image of this target distribution since we are approximating a pullback measure. On the other hand, we see that along the contours of the QoI map defined by the lines $\lambda_1 + \lambda_2 = q$, the distributions are approximately uniform. Since the QoI map cannot distinguish between points belonging to the same contour, the optimization algorithm simply distributes the updated weights equally to all points sampled within a contour. Similar structures are also observed using both the event-based [5] and density-based [6] approaches. This suggests the optimization procedure is producing an approximation to the observation-consistent solution expected from the disintegration theorem in terms of the marginal and family of conditional distributions as described in Section 2.

4. Extensions to Arbitrary Sample-sets. In Section 3, the n samples are assumed to be generated i.i.d. from the initially proposed distribution on the parameter space. Suppose instead we are given a set of n parameter samples $\{\lambda^{(1)}, \ldots, \lambda^{(n)}\}$ and corresponding model evaluations $\{q^{(i)}, \ldots, q^{(n)}\}$ as above but the samples are not necessarily generated from the proposal distribution. Such a situation occurs in practice when analyzing results from a parameter exploration (e.g., a min-to-max sweep) or if we seek to explore alternative proposal distributions but are restricted to a single set of parameter-output samples due to a computational budget. We can use the method in Section 3 to find a set of weights $\{w^{(1)}, \ldots, w^{(n)}\}$ that best approximate the target distribution in the data space from the observed samples. However, the weights computed will then produce the distribution cor-

responding to the pullback measure that is uniquely defined by F^{Λ}_{samp}, where

$$F^{\Lambda}_{samp;n}(\lambda) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}(\lambda^{(i)} \preceq \lambda) \tag{4.1}$$

is the EDF of the sample data not the desired proposal EDF. In other words, this pullback may not have the desired conditional probability structure.

We propose a method that will approximate the desired pullback distribution with conditional probability structure associated with the proposal distribution. First, we use the method described in Section 3 on the *parameter space* to generate a set of initial weights, $\{u^{(1)}, \ldots, u^{(n)}\}$, that solves the constrained minimization problem

$$\begin{array}{rc} \text{minimize} & \frac{1}{2}\left\|F^{\Lambda}_{samp;\boldsymbol{u}}(\lambda) - F^{\Lambda}_{prop}(\lambda)\right\|_2^2 \\ \text{subject to} & \boldsymbol{u} \succeq \mathbf{0} \\ & \mathbf{1}^T\boldsymbol{u} = n, \end{array} \tag{4.2}$$

where we define $F^{\Lambda}_{samp;\boldsymbol{u}}(\lambda)$ as

$$F^{\Lambda}_{samp;\boldsymbol{u}}(\lambda) = \frac{1}{n}\sum_{i=1}^{n} u^{(i)}\mathbb{I}(\lambda^{(i)} \preceq \lambda) \tag{4.3}$$

The constraints in (4.2) are included to ensure that empirical importance weights define a probability measure. As in Section 3, the minimization problem is strictly convex and has a unique solution.

Propagating these samples into the data space, the weighted EDF corresponding to these samples is

$$F^{\mathcal{D}}_{samp;\boldsymbol{u}}(q) = \frac{1}{n}\sum_{i=1}^{n} u^{(i)}\mathbb{I}(q^{(i)} \preceq q). \tag{4.4}$$

The goal is to incorporate this initial set of weights into a second optimization problem that seeks a new set of empirical weights $\{w^{(1)}, \ldots, w^{(n)}\}$ solving the following constrained minimization problem:

$$\begin{array}{rc} \text{minimize} & \frac{1}{2}\left\|F^{\mathcal{D}}_{samp;(\boldsymbol{u},\boldsymbol{w})}(q) - F^{\mathcal{D}}_{targ}(q)\right\|_2^2 \\ \text{subject to} & \boldsymbol{w} \succeq \mathbf{0} \\ & \mathbf{1}^T\boldsymbol{w} = n, \\ & \boldsymbol{u}^T\boldsymbol{w} = n. \end{array} \tag{4.5}$$

where we define the weighted empirical distribution function $F^{\mathcal{D}}_{samp;(\boldsymbol{u},\boldsymbol{w})}$ as

$$F^{\mathcal{D}}_{samp;(\boldsymbol{u},\boldsymbol{w})}(q) = \frac{1}{n}\sum_{i=1}^{n} u^{(i)}w^{(i)}\mathbb{I}(q^{(i)} \preceq q). \tag{4.6}$$

The third constraint $\boldsymbol{u}^T\boldsymbol{w} = n$ ensures that the weights $\{u^{(1)}w^{(1)}, \ldots, u^{(n)}w^{(n)}\}$ define a probability distribution. While the second constraint, $\mathbf{1}^T\boldsymbol{w} = n$, is the subject of ongoing research, we motivate its inclusion. First observe that removing this constraint implies that the optimization problem can be equivalently rephrased as optimizing for the modified weight vector $\tilde{\boldsymbol{w}}$ where the ith component is $u^{(i)}w^{(i)}$. This is subsequently equivalent to

solving the previous optimization problem assuming i.i.d. samples from the proposal, which is fundamentally incorrect in this case. The impacts of this are best understood after discussing the associated solution to the inverse problem. As in Section 3, we will use the multiplicative weights $\{u^{(1)}w^{(1)}, \ldots, u^{(n)}w^{(n)}\}$ computed on $\mathcal{D}$ to define a weighted EDF $F^{\Lambda}_{samp;(\boldsymbol{u},\boldsymbol{w})}$ on the parameter space, where

$$F^{\Lambda}_{samp;(\boldsymbol{u},\boldsymbol{w})}(\lambda) = \frac{1}{n}\sum_{i=1}^{n} u^{(i)}w^{(i)}\mathbb{I}(\lambda^{(i)} \preceq \lambda). \tag{4.7}$$

We conjecture that under certain conditions on the sample generating distribution, coupled with the constraint $\mathbf{1}^T\boldsymbol{w} = n$, that $F_{samp;(\boldsymbol{u};\boldsymbol{w})}$ converges to a pullback of the target distribution with conditional probabilities described by the proposal distribution. Note that if the constraint, $\mathbf{1}^T\boldsymbol{w} = n$, is not included in the optimization problem (4.5), then this suggests that $F_{samp;(\boldsymbol{u},\boldsymbol{w})} = F_{samp;\tilde{\boldsymbol{w}}}$ will converge to a pullback distribution with conditional probabilities on generalized contours given by the sample, not the proposal, distribution.

Following the derivation in [1], we can assume the support of the samples is confined to the unit hypercube and expand the objective function in (4.5) as

$$\begin{aligned}\frac{1}{2}\left\|F^{\mathcal{D}}_{samp;(\boldsymbol{u},\boldsymbol{w})}(q) - F^{\mathcal{D}}_{targ}(q)\right\|_2^2 =& \frac{1}{2}\int_0^1\cdots\int_0^1\left(\frac{1}{n}\sum_{i=1}^{n}u^{(i)}w^{(i)}\mathbb{I}(q^{(i)}\preceq q) - F^{\mathcal{D}}_{targ}(q)\right)^2 dq\\ =& \frac{1}{2}\int_0^1\cdots\int_0^1\left[\left(\frac{1}{n}\sum_{i=1}^{n}u^{(i)}w^{(i)}\mathbb{I}(q^{(i)}\preceq q)\right)^2\right.\\ &\left. - \frac{F^{\mathcal{D}}_{targ}(q)}{n}\sum_{i=1}^{n}u^{(i)}w^{(i)}\mathbb{I}(q^{(i)}\preceq q) + (F^{\mathcal{D}}_{targ}(q))^2\right] dq.\end{aligned} \tag{4.8}$$

As in [1], the third term in the integrand is independent of $\boldsymbol{w}$ so it can be discarded. We consider the first two terms of the optimization statement. The first is equal to

$$\begin{aligned}\frac{1}{2}\int_0^1\cdots\int_0^1\left(\frac{1}{n}\sum_{i=1}^{n}u^{(i)}w^{(i)}\mathbb{I}(q^{(i)}\preceq q)\right)^2 &= (\mathrm{diag}(\boldsymbol{u})\boldsymbol{w})^T H\,\mathrm{diag}(\boldsymbol{u})\boldsymbol{w}\\ &= \frac{1}{2}\boldsymbol{w}^T DHD\boldsymbol{w},\end{aligned} \tag{4.9}$$

where $D = \mathrm{diag}(\boldsymbol{u})$, H is defined as the Hadamard product $H^1 \circ \cdots \circ H^d$, and

$$H^k_{i,j} = \int_{z_k^{(i,j)}}^1 dq_k, \tag{4.10}$$

for $k \in \{1, \ldots, d\}$, where $z_k^{(i,j)} = \max(q_k^{(i)}, q_k^{(j)})$ and $q_k^{(i)}$ is the kth entry of $q^{(i)}$. The second term in the integrand can be rewritten as

$$\frac{1}{2}\int_0^1\cdots\int_0^1\frac{2F^{\mathcal{D}}_{targ}(q)}{n}\sum_{i=1}^{n}u^{(i)}w^{(i)}\mathbb{I}(q^{(i)}\preceq q)dq = (D\boldsymbol{w})^T\boldsymbol{b} = \boldsymbol{w}^T D\boldsymbol{b}, \tag{4.11}$$

where the ith entry of $\boldsymbol{b}$ is

$$b_i = \frac{1}{n}\int_{q_1^{(i)}}^1\cdots\int_{q_d^{(i)}}^1 F^{\mathcal{D}}_{targ}(q)dq. \tag{4.12}$$

Thus we have rewritten the optimization problem Eq. (4.5) as

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}\left(\boldsymbol{w}^T DHD\boldsymbol{w} - 2\boldsymbol{w}^T D\boldsymbol{b}\right) \\ \text{subject to} & \boldsymbol{w} \succeq \mathbf{0} \\ & \boldsymbol{u}^T\boldsymbol{w} = n, \\ & \mathbf{1}^T\boldsymbol{w} = n. \end{array} \tag{4.13}$$

Note that by fixing $\boldsymbol{w} = \mathbf{1}^T$ we recover the original formulation, so the same implementation can be used to solve either problem.

The Lagrangian of the optimization statement is

$$\mathcal{L}(\boldsymbol{w}, \boldsymbol{\delta}, \boldsymbol{\nu}) = \frac{1}{2}\left(\boldsymbol{w}^T DHD\boldsymbol{w} - 2\boldsymbol{w}^T D\boldsymbol{b}\right) + \boldsymbol{\delta}^T\left(\begin{bmatrix}\boldsymbol{u}^T \\ \mathbf{1}^T\end{bmatrix}\boldsymbol{w} - \begin{bmatrix} n \\ n\end{bmatrix}\right) - \boldsymbol{\nu}^T\boldsymbol{w}, \tag{4.14}$$

where $\boldsymbol{\delta}$ and $\boldsymbol{\nu}$ are vectors of Lagrange multipliers of size n. The optimal solution $\hat{\boldsymbol{w}}$ then satisfies the KKT conditions:

$$\begin{aligned} \frac{\partial\mathcal{L}(\hat{\boldsymbol{w}}, \boldsymbol{\delta}, \boldsymbol{\nu})}{\partial \boldsymbol{w}} &= \mathbf{0} = DHD\boldsymbol{w} - D\boldsymbol{b} + \boldsymbol{\delta}^T\begin{bmatrix}\boldsymbol{u}^T \\ \mathbf{1}^T\end{bmatrix} - \boldsymbol{\nu}, \\ &\hat{\boldsymbol{w}} \succeq \mathbf{0}, \\ &\boldsymbol{\nu} \succeq \mathbf{0}, \\ &\begin{bmatrix}\boldsymbol{u}^T \\ \mathbb{1}^T\end{bmatrix}\hat{\boldsymbol{w}} = \begin{bmatrix} n \\ n\end{bmatrix}, \\ &\boldsymbol{\delta} \text{ is sign unrestricted}, \\ &\nu^{(j)}\hat{w}^{(j)} = 0 \quad \forall j \in \{1, \ldots, n\}. \end{aligned} \tag{4.15}$$

H is symmetric positive definite by definition. If D is a positive diagonal matrix, then DHD is symmetric positive definite and thus the problem 4.13 is a strictly convex quadratic program with linear constraints and thus admits a global solution [3]. While D is guaranteed to be at least positive semi-definite by the constraint in Eq. 4.2 that $\boldsymbol{u} \succeq \mathbf{0}$, it may have zero entries if $u^{(j)} = 0$ for some j. For now, we assume that D is positive definite, or that there are no zero weights in the first optimization, thus the problem is a strictly convex quadratic program and admits a global solution.

4.1. An illustrative example. We illustrate the concept with a similar example as in Section 3.1, but here the samples are drawn uniform in each direction, the proposal distribution is uniform $\mathcal{U}(-1, 1)$ in the λ_2 direction of the parameter space, and normal $\mathcal{N}(0.1, 0.2)$ in the λ_1 direction. The target distribution is $\mathcal{N}(0.1, 0.08)$. The mapping Q and proposal and target distribution remain the same. The weights for the first optimization on the parameter space, a straightforward application of the method in [1], is shown in Figure 4.1. The normal structure in the λ_1 direction has been recovered, maintaining uniformity in the λ_2 direction.

After propagating these samples and weights through the model and performing the second optimization, Figure 4.2 shows the resulting weighted EDF in the data space, which we can see matches the target distribution well. The resulting weights also appear to preserve the desired structure on the parameter space well, as in Section 3 we can see the structure from both the proposal distribution and the observed distribution in each direction.

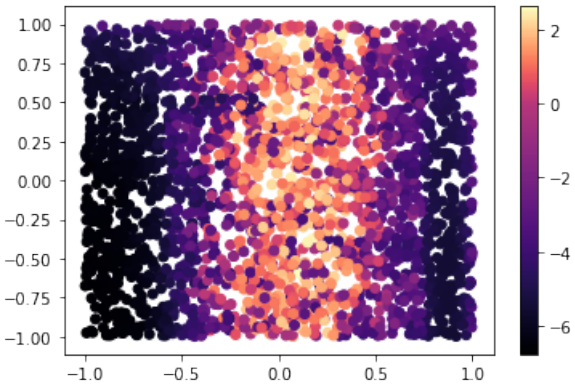


Fig. 4.1: Log of initial weights on Λ, $n = 2500$

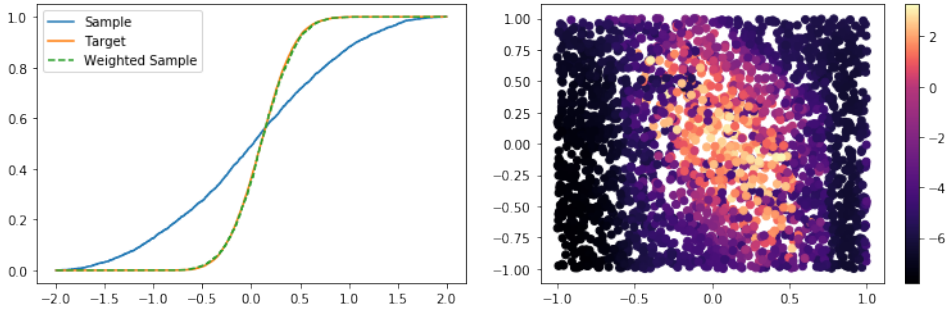


Fig. 4.2: Distributions on $\mathcal{D}$ and log of weights for $F^{\Lambda}_{prop;\boldsymbol{w}}$ on Λ ($n = 2500$).

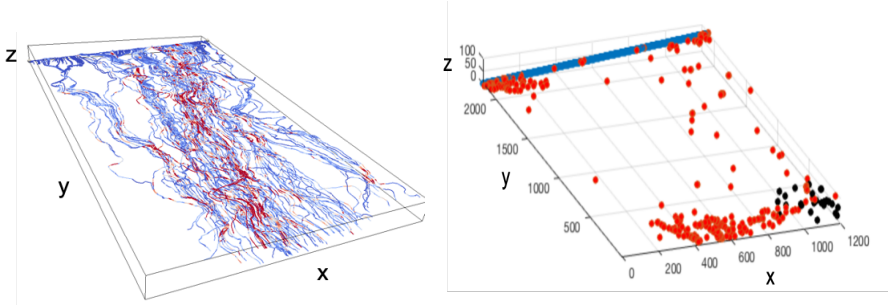
5. Applications. In this section, we consider two applications of interest to Sandia National Labs, as well as the Department of Energy, Office of Science. The first application is a physics-based model for steady-state single phase flow in porous media where each evaluation of the model requires nontrivial computational resources and the goal is to invert based on observed tracer data. The second application is relevant to scientific machine learning where we are given a previously trained neural network and some output data (either from training or testing) and the goal is to infer a distribution on the inputs to the model that could have generated this data.

5.1. Tracers from single-phase flow in porous media. We first consider a physics-based model for single phase incompressible flow in porous media. The model represents 3-dimensional flow in a fluvial reservoir where the permeability field is based on the SPE10 dataset and a pressure gradient driving the flow is induced using Dirichlet conditions imposing a pressure drop from one side to the other.

The computational domain is $\Omega = [0, 1200] \times [0, 2200] \times [0, 100]$ (unit are ft) and the permeability is defined on a uniform $60 \times 220 \times 50$ rectangular grid. While the SPE10 data set comes with x-, y-, and z-permeabilities, we only utilize the x-permeability to create an isotropic tensor. Since locally conservative velocities are required for most applications in porous media, we use a hybridized mixed finite element formulation to construct a numerical approximation. A full description of this formulation is beyond the scope of this paper, but

we refer the interested reader to [4] for more details.

ParaView allows us to create tracers to track the flow through the media from certain starting points. The starting points can be generated as a high resolution line source, we use the line from $(0, 2200, 0)$ to $(1200, 2100, 100)$. The tracers flow through the media from the high pressure area at the lower end of the y direction to the low pressure area at the higher end of the y direction, as shown in Figure 5.1(a). As each tracer flows through the media, it will either stop at some point, or exit the domain.



(a) Tracers for proposal distribution

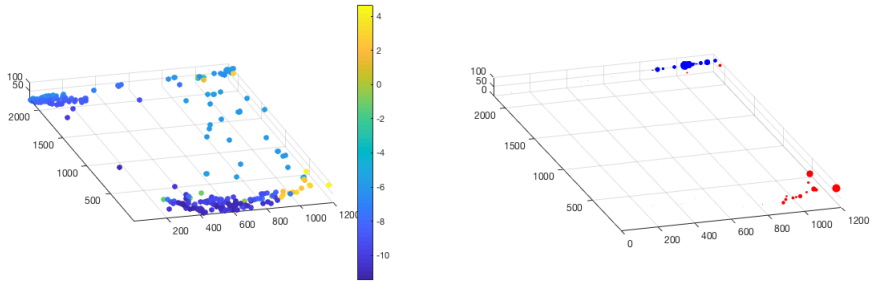
(b) Starting/ending proposal samples, and observed samples

Fig. 5.1: Tracers in porous media

The parameter space Λ is the three-dimensional line source, and the output space is the three-dimensional space $\mathcal{D}$ of all possible ending points from tracers generated along the source line. Given a set of target ending points, as well as a set of proposal starting/ending point pairs, we seek a set of weights on the proposal starting points that approximate a distribution that pushes forward to the empirical distribution defined by the target ending points. In Figure 5.1(b), the starting points are uniformly distributed along the blue line, the ending points are shown in red, and the observed points are shown in black. The proposal distribution is uniform along the source line, and the target distribution is uniformly distributed in a box at the lower right corner of the domain. The results of the weighting on $\mathcal{D}$ are shown in Figure 5.2(a). The proposal ending points that are closest in x-distance to the range of the observation data are weighted heavily and the ending points that are farther away in x-distance are given smaller weights. The weights on the proposal samples in Λ and $\mathcal{D}$ are shown in Figure 5.2(b), represented by the size of the points.

We see that the majority of the probability is assigned to proposal samples where the ending points are close, in the x-direction, to the observations. As further confirmation that our method provides physically meaningful results, we can also perform an experiment to determine which of the parameters (x, y, or z) are important, by rerunning the analysis excluding each direction in turn. From Figure 5.3, we can see that the y and z directions are not informative without the x direction included. This is consistent with our expectations since the flow is primarily in the x-direction.

5.2. Inferring training data from a neural network. In this section, we consider a problem relevant to scientific machine learning where the goal is to infer a distribution on the model inputs given a distribution on model outputs. To that end, we utilize a previously generated dataset containing 10,000 input-output pairs, where the input space is 100-dimensional and the output space is 3-dimensional and the mapping is nonlinear. We then use PyTorch to construct a neural network surrogate approximation of the mapping.



(a) Weights on proposal end points (log scale) (b) Weights on proposal start/end points, larger points correspond to larger weights

Fig. 5.2: Weighted tracer start/end points for porous media

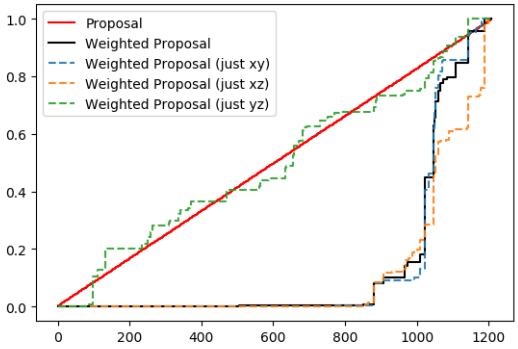
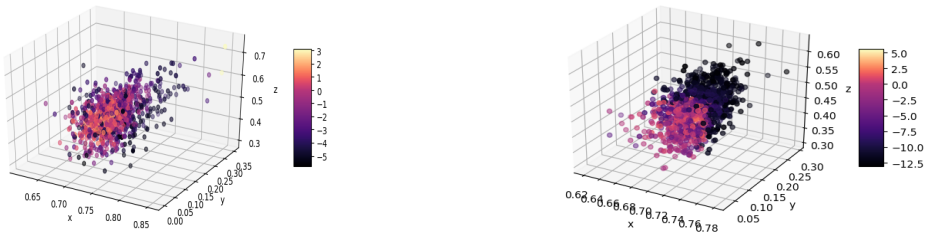


Fig. 5.3: Weighted EDFs in direction of increasing magnitude on source line

We utilize a network with 5 hidden layers with 80, 60, 40, 20, 10 nodes respectively. The network is trained using the first 9000 input-output pairs in the dataset and tested against the remaining 1000 pairs. To avoid randomness in the network due to the utilization of the stochastic gradient descent algorithm or random starting points for the training/optimization of the neural network, we train the network once, save it and then treat it as a fixed surrogate model. This is consistent with this idealized scenario where we assume we are given a previously trained neural network and seek to use this as a surrogate model to solve a stochastic inverse problem. We are aware that errors in the surrogate model can lead to errors in the updated distribution, and this was studied extensively in [7] for the density-based approach, but we leave this issue for future work for the EDF-based approach.

The observed data for our inverse problem is the set of outputs generated from the training and testing data. Our goal is to both determine a set of weights on the input samples that define a pullback measure for the distribution defined by the observed data, and to test if we can infer any information about the data-generating distribution of the input training data, which is normally distributed in each direction with mean 0 and variance

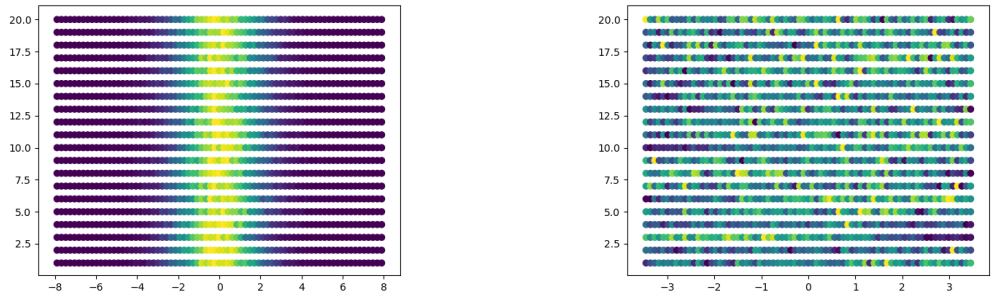
1. We show results from two different proposal distributions on the input space. The first is for a normal distribution (in each parameter) centered at 0 with standard deviation 1.5. The resulting weights on the data space are shown in Figure 5.4(a). The resulting weights on the data space for a uniform distribution from -2 to 2 on each parameter in the input space and are shown in Figure 5.4(b).



(a) Normal proposal (b) Uniform proposal

Fig. 5.4: Weights on $\mathcal{D}$, machine learning application (log-scale)

Using plot of weights or empirical distribution functions on the input space to infer qualitative information is challenging task on a 100-dimensional space. We can glean some useful information by estimating the marginal probability densities that correspond to the marginal weighted EDFs in various input directions. For the normal proposal, we can see from Figure 5.5(a) that the marginals do not change much from the proposal distribution and maintain normal structure - indicating that the pullback distribution is roughly equivalent to the data-generating distribution. For the uniform proposal, the marginals estimated are very different, as shown in Figure 5.5(b). Although the pullback measure pushes forward through the model to match the observed distribution, the pullback marginals are not the same as the data-generating distribution.



(a) Normal proposal (b) Uniform proposal

Fig. 5.5: Marginal densities in the first 20 parameter directions

6. Conclusions and Future Work. In this work, we have developed an optimization-based approach for solving observation-consistent stochastic inverse problems. Previous approaches to solving observation-consistent inversion problems require approximations of events or densities. We show that we are able to extend the observation-consistent framework using the optimization-based approach method to cases with limited observational data, when densities and events cannot be approximated effectively. We have also extended the solution to problems with arbitrary sampling when we do not have i.i.d. samples from the proposal distribution. This is an extension to both the observation-consistent framework and the optimization-based weighted empirical distribution approach developed in [1].

In future work, we aim to formalize a theoretical proof of the stability of the pullback measure that is defined by the weighted EDF on the parameter space, as well as a theoretical proof to justify the generalization of the method to the case of arbitrary sample sets. We are also interested in quantifying the effect of the model form errors on the weighted EDFs and the corresponding approximations to the pullback measure. Future directions will also include a utilization of this approach in the context of optimal experimental design and an exploration of the use of this approach to efficiently perform hierarchical Bayesian inference.

REFERENCES

[1] S. Amaral, D. Allaire, and K. Willcox, *Optimal L_2-norm empirical importance weights for the change of probability measure*, Statistics and Computing, 27 (2017), pp. 625–643.
[2] M. S. Andersen, J. Dahl, and L. Vandenberghe, *Cvxopt: A python package for convex optimization, version 1.2.5*, 2020.
[3] S. Boyd and L. Vandenberghe, *Convex Optimization*, Cambridge University Press, Cambridge, 2004.
[4] F. Brezzi and M. Fortin, *Mixed and hybrid finite element methods*, vol. 15 of Springer Series in Computational Mathematics, Springer-Verlag, New York, 1991.
[5] T. Butler, D. Estep, S. Tavener, C. Dawson, and J. J. Westerink, *A measure-theoretic computational method for inverse sensitivity problems iii: Multiple quantities of interest*, SIAM Journal on Uncertainty Quantification, 2 (2014), p. 174–202.
[6] T. Butler, J. Jakeman, and T. Wildey, *Combining push-forward measures and bayes' rule to construct consistent solutions to stochastic inverse problems*, SIAM Journal on Scientific Computing, 40 (2018), pp. A984–A1011.
[7] T. Butler, J. Jakeman, and T. Wildey, *Convergence of probability densities using approximate models for forward and inverse problems in uncertainty quantification*, SIAM J. Sci. Comput., 40 (2018), pp. A3523–A3548.
[8] J. Chang and D. Pollard, *Conditioning as disintegration*, Statistica Neerlandica, 51 (1997), pp. 287–317.

CONCENTRIC SPHERICAL GNN FOR 3D REPRESENTATION LEARNING

JAMES FOX*, SIVASANKARAN RAJAMANICKAM†, AND LE SONG‡

Abstract. Learning 3D representations that generalize well to arbitrarily oriented inputs is a challenge of practical importance in applications varying from computer vision to physics and chemistry. We propose a novel multi-resolution convolutional architecture for learning over concentric spherical feature maps, of which the single sphere representation is a special case. Our hierarchical architecture is based on alternatively learning to incorporate both intra-sphere and inter-sphere information. We show the applicability of our method for two different types of 3D inputs, mesh objects, which can be regularly sampled, and point clouds, which are irregularly distributed. We also propose an efficient mapping of point clouds to concentric spherical images using radial basis functions, thereby bridging spherical convolutions on grids with general point clouds. We demonstrate the effectiveness of our approach in achieving state-of-the-art performance on 3D classification tasks with rotated data.

1. Introduction. While convolutional neural networks have been applied to great success to 2D images, extending the same success to geometries in 3D has proven more challenging. A desirable property and challenge in this setting is to learn descriptive representations that are also equivariant to any 3D rotation. [3] and [5] showed that the spherical domain permits learning such rotationally equivariant representations, by defining convolutions with respect to spherical harmonics. In practice, 3D convolutions are implemented via discretization of the sphere. Earlier spherical CNNs used spherical coordinate grids, but these discretizations result in non-uniform samplings of the sphere, which is non-ideal. Furthermore, spherical convolutions defined on these grids scale with $O(N^{1.5})$ complexity (N as the number of grid points). Subequent works, [6], [2], [4], designed more scalable $O(N)$ convolutions focusing on more uniform spherical discretizations.

Existing spherical CNNs operate over a spherical image, resulting from projection of data to a bounding sphere. We show that is more expressive and general to instead operate over a concentric, multi-spherical discretization for representing data. Our main innovation is introducing a new two-phase convolutional scheme for learning over a concentric spheres representation, by alternating between inter-sphere and intra-sphere convolutional blocks. We use graph convolutions to incorporate inter-sphere information, and 1D convolutions to incorporate radial information. Similar to [6],[2], we focus on the icosahedral spherical discretization, which produces a mostly regular sampling over the sphere. Our proposed architecture is hierarchical, following the recursive coarsening hierarchy of the icosahedron. Combinining intra-sphere and inter-sphere convolutions has a conceptual analogy to gradually incorporating information over volumetric sectors. At the same time, our learned representation retains rotational equivariance from scalable graph-based spherical convolutions.

We demonstrate the effectiveness and generality of our approach through two 3D classification experiments with different types of input data: mesh objects and general point clouds. The latter poses an additional challenge for discretizations, as native point clouds are non-uniformly distributed in 3D space.

To summarize our contributions:

1. We propose a novel multi-sphere icosahedral discretization for representation of 3D data, and show that incorporating the radial dimension can greatly enhance representation ability over single-sphere representations.

*Georgia Institute of Technology, jfox43@gatech.edu
†Sandia National Laboratories,srajama@sandia.gov
‡Georgia Institute of Technology, lsong@cc.gatech.edu

2. We introduce a novel convolutional architecture for multi-sphere discretization by introducing two different types of convolutions, conceptually separated as intra-sphere and inter-sphere. Combining graph convolutions (intra-sphere) with 1D radial convolutions (inter-sphere) leads to an expressive architecture that is also rotationally equivariant. Our proposed convolutions are also scalable, each being linear with respect to total grid size.
3. We design new mappings for both 3D mesh objects and general point clouds to the proposed representation. We achieve state-of-art performance on ModelNet40 point cloud classification, using the proposed model and a data mapping using radial basis functions. We also improve on existing Spherical CNN performance in SHREC17 3D mesh classificaiton task by utilizing multi-radius information.

2. Related Work.

2.1. Spherical CNNs. The goal of learning rotationally invariant representations of 3D geometries has led to several ideas for rotationally equivariant convolutions in the spherical domain. [3], [5] defined spherical convolutions that are rotationally equivariant to rotations of the $SO(3)$ group. However, these convolutions are restricted to non-uniform grid samplings and scale superlinearly with respect to grid resolution. Later works have explored more scalable convolutions on other spherical discretizations, achieving linear complexity with respect to grid resolution.

[6] proposed using parameterized differential operators to form convolutional kernels over the icosahedron, where equivariance is restricted to rotations about the z-axis. [2] proposed gauge equivariant convolutions on manifolds, operating on feature fields corresponding to underlying geometric entities. This was applied to achieve rotationally equivariant convolutions over the icosahedral discretization. [4] propose a graph convolution-based spherical CNN using spectral filters, along with a distance-weighted nearest-neighbors graph construction scheme that allows balancing between rotational equivariance and efficiency, when applied to different types of grids.

Other spherical CNNs have been designed in the context of handling arbitrary point cloud data, which typically requires first mapping the data to a discretization. [11] uses graph-convolution inspired message passing oeprators for learning over the icosahedral discretization. Our work is similar to [11] and [4]) in terms of using graph-based spherical convolutions, but we generalize to multi-sphere convolutions. [16] is the most related work in terms of multi-sphere representation learning. The authors propose a spherical voxel grid, and extending the SO3 convolutions of [3] to incorporate the radial dimension. Our work treats spherical and radial convolutions as distinct, which observes much better results in practice. We also use more scalable spherical convolutions defined on the uniform icosahedral grid.

2.2. Pointwise Convolution Networks. There is a significant body of work on learning point cloud representations using pointwise convolutions, beginning with with [10] which proposed learning permutation invariant functions that directly operate on point coordinates. Only more recently have such methods have been developed towards learning rotationally invariant representations.

[14] and [9] both propose pointwise convolutional filters based on spherical harmonic functions to achieve rotational equivariance (or invariance). Distance information is recorded through learned functions in the former, and radial sampling in the latter. While these filters are defined with respect to all-to-all convolution between points, in practice convolutions are limited to k-nearest neighbors ([9]) for scalability. [1], [13], [17] all extract rotationally invariant features (i.e. low-level geometric features such as angles and distances) from the

point cloud as input to their respective convolutional architectures. These features are hand-engineered based on carefully picking local frames of references, or global in the case of [13].

3. Representation by Concentric Spheres. Existing work on spherical CNNs operate on spherical grids, where data is typically projected to and defined on grid points. However, projecting 3D data to a single sphere may not always be sufficient or appropriate. Simple projections may be lossy when describing highly non-convex shapes, for instance if the shape curves in on itself. To increase capacity to distinguish different data distributions, we introduce a new discretization based on concentric spheres. The introduction of concentric spheres additionally discretizes 3D space in the radial dimension. Single sphere discretization is a special case in our proposed paradigm, corresponding to an outermost bounding radius.

Spherical Discretization. We work with an icosahedral grid discretization of the sphere, as it (1) has a regular (recursive) construction and (2) results in a largely uniform sampling over the sphere. The base icosahedron I_0 has 12 vertices, forming 20 equilateral triangle faces (each face with 3 edges). Each vertex is incident with 5 triangles. It can be recursively refined by subviding each triangle into 4 smaller ones, with the number of vertices scaling by level as $|V| = 10 * 4^l + 2$. See Fig. 3.2 for icosahedron illustration.

Radial Discretization. We construct a multi-radius discretization for R concentric spheres by stacking stacking R icosahedral grids. The same spherical discretization is shared across concentric spheres, enabling efficient convolutions. Assuming unit radius normalization, we use a uniform discretization that results in concentric spheres scaled to radiuses $[\frac{1}{R}, \frac{2}{R}, ..., 1]$.

Intra-sphere Convolutions.

There is a growing body of work addressing design of rotation-equivariant filters over spherical feature maps. We focus on graph convolutional filters for intra-sphere convolutions, as graph convolutions are scalable and lead to equivariant representations, up to discretization effects [4]. While there is a rich body of work on graph-based convolutions and its variants, this work uses the graph convolution from [7].

This motivates our construction of the undirected graph $G^{(l)} = (V^{(l)}, E^{(l)})$ from a level l icosahedron I_l. Vertices of the vertex set $V^{(l)}$ correspond one-to-one with vertices of I_l projected to unit sphere. $E^{(l)}$ is simply the set of all (bidirectional) face edges of the icosahedron (projected to unit sphere). Each vertex has a degree of either 5 or 6 in this construction. Since each edge corresponds to approximately equal distance between two points of the sphere [15], G_l is also treated as an unweighted graph.

Let t index layer, $i \in [0, R-1]$ index the radial dimension, and $u \in [0, V-1]$ index the vertices. Also let Z be parameters, g be feature vectors, and σ indicate nonlinear activation function. The intra-sphere convolution output $h_{i,u}$ is given by Eq. 3.1, where $N(u)$ indicates adjacent neighbors of vertex u (including u itself).

$$g_{i,u}^{(t+1)} = \sigma\Big(\sum_{v \in N(u)} \frac{1}{\sqrt{d_u d_v}} Z g_{i,v}^{(t)}\Big) \tag{3.1}$$

The overall complexity of intra-sphere convolution is equivalent to the total grid size $O(N)$, where $N = R * V^{(l)}$. In practice, R can also be restricted to a relatively small constant.

Inter-sphere Convolutions. We introduce *radial convolutions* to incorporate inter-sphere information, implemented as the 1D convolutions over multiple radial representations of each vertex. Importantly, radial convolutions are also rotationally invariant, as 1D convolution operates over channels of the same vertex. See Fig. 3.1 for illustration of radial and graph convolutions with respect to concentric spheres representation.

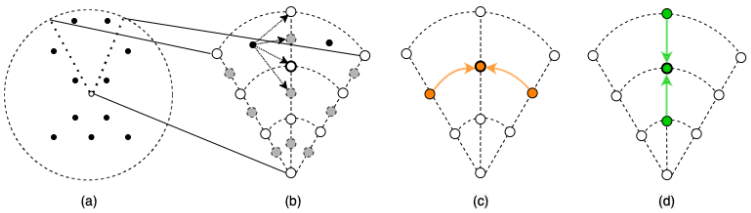


Fig. 3.1: 2D cross-section illustrations of concentric spheres representation. (a) shows an example point cloud with respect to bounding sphere. (b) zooms in on a particular sector. Data points are mapped to RBF values defined on vertices, where an expanded discretization is used. (c) Intra-sphere convolution and (d) inter-sphere convolutions are are applied to the target vertex (bolded). In reality, intra-sphere convolution on local neighborhood of sphere, and involves 5 or 6 vertices.

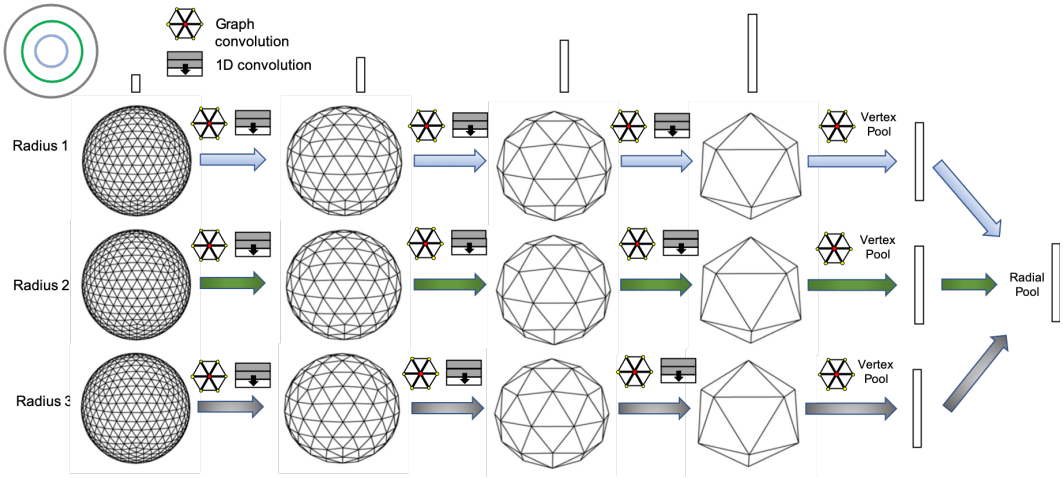


Fig. 3.2: Example multi-radius architecture with $R = 3$ concentric spheres. Graph convolutions, followed by radial convolutions, are applied over a sequence of discretization levels. Pooling coarsens the discretization to a lower level. Vertex-wise and radial-wise pooling is applied to obtain a final representation for classifier. Icosahedron visualization from [12].

We introduce additional notation to describe radial convolutions. Let Z be parameters, and h feature vectors. Then the inter-sphere convolution output $g_{i,u}$ convolution is given by Eq. 3.2. K indicates 1D kernel size. We assume K to be odd valued, and pad inputs in the radial dimension to ensure dimension R is maintained across convolutions.

$$h_{i,u}^{(t+1)} = \sigma\Big(\sum_{k=-\lfloor \frac{K}{2} \rfloor}^{\lfloor \frac{K}{2} \rfloor} W_{k+\lfloor \frac{K}{2} \rfloor} h_{i+k,u}^{(t)}\Big) \tag{3.2}$$

The overall complexity of intra-sphere convolution is also $O(N)$. Intra-sphere convolutions require Z parameters and inter-sphere convolutions require $K * W$ parameters.

Concentric Spherical GNN Architecture. Fig. 3.2 gives an example illustration of an end-to-end architecture using both convolutions. Importantly, radial convolution blocks are introduced alongside graph convolutional blocks at every level of the spherical discretization hierarchy, to incorporate inter-sphere information gradually. From a icosahedron of level L refinement, we construct a sequences of graphs $[G_L, G_{L-1}, ..., G_0]$. Each G_l

carries an additional R dimension, corresponding to spheres at different radial levels. Each level l features two blocks of convolutions: graph convolutions, followed by radial convolution. These correspond to intra-sphere and inter-sphere convolutions respectively. This is followed by vertex neighborhood pooling, which downsamples the graph from G_l to G_{l-1}. The size of the radial dimension remains constant, until final pooling.

4. Point Cloud to Concentric Spherical Signal. We consider the problem of mapping a point cloud $P \in \mathbb{R}^{N\times 3}$ point cloud to an initial spherical feature map $M \in \mathbb{R}^{R\times V\times F}$, where F is number of input channels. While the concentric grid representation is defined discretely at fixed positions, the space of data point locations is continuous. We also aim to capture the distribution of points in a continuous way. To do so, we summarize the contribution of points using the Gaussian radial basis function:

$$f(x) = \sum_{j=1}^{N} \phi(||x - p_j||_2^2) \tag{4.1}$$

Here N is the number of data points, and $\phi = \exp(-\gamma r^2)$, parameterized by the bandwidth γ. In practice we limit computation to a local neighborhood (instead of considering all points), and choose γ accordingly. See Sec. A for additional details.

One possible mapping is to compute Eq. 4.1 at every vertex position $x_{i,u} : i \in [0, R-1], u \in V$ of the spherical discretization, resulting in a single channel feature map. However, it is possible to obtain better resolution in capturing distribution of surrounding points by further sub-diving the discretized space (taking inspiration from [8]) along the radial dimension. Subdividing along radial dimension by a factor K_e results in a new spherical discretization with size $R' = R * K_e$ in the radial dimension. We compute the RBF value at every vertex position of this new discretization, resulting in a feature map of dimension of $M' \in \mathbb{R}^{R'\times V\times 1}$. We then map back to the original discretization by assigning $M_{i,u} = [M'_{j,u} : \ j \in (iK_e, iK_e + 1, ..., 2iK_e, 2iK_e + 1, ..., 3iK_e - 1)]$, resulting in a $[R \times V \times 2 * K_e]$ feature map. In other words, to better capture data distribution, multiple RBF values are assigned to each vertex by further sub-dividing space in the radial dimension.

5. Experiments.

5.1. ModelNet40 Point Cloud Classification. We consider the ModelNet40 3D shape classification task, with 12308 shapes and 40 classes. Following convention, each point cloud has 1024 points. For all experiments, 9840 shapes are used for training and 2468 for testing.

Architecture and Hyperparameters. Figure 5.1 shows a complete architecture overview. Point clouds are first mapped to 16 concentric spheres with level 4 icosahedral discretization ($L = 4, R = 16$), using RBF kernels with threshold $T = 0.01$. 1D convolutions use a kernel size of 3. Each graph and 1D convolution is followed by batch normalization and ReLU as nonlinear activation. Additionally, skip connections are added between every graph convolution layer, whenever input dimension matches output. The model is trained with Adam optimizer for 30 epochs using initial learning rate of 1e-3, along with learning rate decay by 0.1 at 15 and 25 epochs. The batch size is 32. Each training epoch took 5 minutes, while inference (of validation set) took 7-8 seconds. Data loading and transformation times are included.

Results. We present our results and compare against other related works in Table 5.1, in four different train/test data orientation settings. When training with rotations, a new rotation is sampled per instance in each epoch. Rotation is the only augmentation used in comparisons. We report accuracy as average validation score across last 5 epochs of training, due to lack of standard validation/test split and to account for variation.

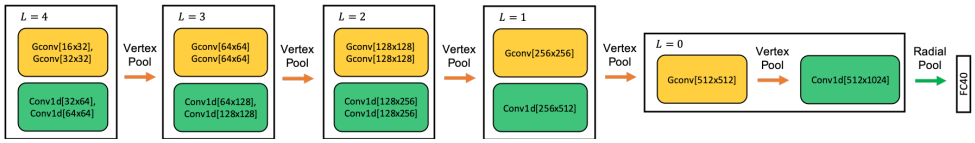


Fig. 5.1: Architecture for ModelNet40 classification. Input dimension is 16, resulting from point cloud RBF data mapping. "Gconv" is graph convolution applied over graph connectivity of the sphere. "Conv1d" is 1D convolution, applied over the radial dimension. L denotes discretization level, as the representation is coarsened following each vertex pooling step. A final pooling of radial dimension results in a 1024 dimensional vector.

Table 5.1: ModelNet40 classification results, across four train/test data orientation settings. NR denotes original data (no rotations), z is arbitrary rotation about z axis, and $SO3$ is arbitrary rotation. For example, SO3/SO3 means training and testing with arbitrary rotations of the data.

Method	Input	z/z	z/SO3	NR/SO3	SO3/SO3
Pointwise Convolution					
PointNet [10]	xyz	0.875	0.229	0.081	0.849
ClusterNet [1]	xyz	0.871[1]	0.871[1]	*[2]	0.871[1]
RIConv [17]	xyz	0.865[1]	0.864[1]	*[2]	0.864[1]
SPHNet [9]	xyz	0.865	0.856	0.854	0.870
SRINet [13]	xyz+normal	*[2]	*[2]	**0.870**[1]	*[2]
Spherical CNN					
SFCNN [11]	xyz	**0.888**	0.831	0.350	0.874
PRIN [16]	xyz	0.819	0.765	0.753	0.810
Ours (CSGNN)	xyz	0.884	**0.874**	0.833	**0.884**

Our method achieves state-of-the-art results in z/SO3 and SO3/SO3 settings, i.e. testing on arbitrarily rotated data. For more detailed comparison, we loosely categorize compared works by method into two categories: pointwise convolution networks and spherical CNNs. Methods in the former category operate directly on data points in 3D space, while methods in the latter operate on a spherical discretization. Our work is most closely related to methods in the spherical CNN category.

Similar to our work, PRIN also explored learning a concentric spherical representation based on extending SO3 convolutions from [3]. Our method is based on separate graph and radial convolutions, which achieves much better performance in all settings. SFCNN has similarity to our work using graph convolution-inspired message passing filters, and hierarchically learning over the icosahedral discretization of the sphere. However, SFCNN is restricted to a single-sphere representation, and also relies on a PointNet-like learned module to project points to spherical features. While this learned projection should be able capture some degree of multi-radius information in the point distribution, best results seem to be achieved by learning from both intra-sphere and inter-sphere convolutions. Compared to our approach, SFCNN also has a relatively higher performance gap between z/z and any

[1] Using numbers reported in respective authors' paper.

[2] ClusterNet, RIConv, and SRINet are designed to be strictly rotationally invariant. Nearly identical performance is expected across all settings (including settings not reported by respective authors).

SO3 test setting (significantly so for z/SO3 and NR/SO3), which suggests greater difficulty achieving rotational invariance. CSGNN, similar to SFCNN and PRIN, exhibits some drop in performance in the z/SO3 and NR/SO3 settings. This could partly be due to effects of discretization and data mapping. However, this gap is relatively smallest in CSGNN, and there is essentially no gap in z/z and SO3/SO3 performance.

ClusterNet, RIConv, and SRINet use hand-crafted rotationally invariant geometric features as inputs, and so there is negligble to no performance gap in testing with or without rotations. By contrast, our method largely learns to extract features directly from the input (outside of an initial step mapping points to vertices). PointNet, unlike the other baselines, was not designed to be rotationally equivariant. This reflects in the relatively significant difference in test performance with and without rotations. Even when training with rotations, and using a learned alignment module that attempts to learn a canonical transformation, SO3/SO3 performance is not competitive with that of most other baselines.

5.2. SHREC17 3D Shapes. The SCHREC17 task has 51300 3D models and 55 categories. We use the version where all models have been randomly perturbed by rotations. Here the inputs are not point clouds, but mesh objects. [3],[5] presented a ray-casting scheme to regularly sample information incident to outermost mesh surfaces and obtain features maps defined over the spherical discretization. For sufficiently non-convex mesh objects, a single sphere projection may result in information loss. For example, when a ray is incident to multiple surfaces occurring at different radii, this information is discarded by existing methods. We propose a new data mapping that generalizes single sphere representation to a concentric spherical representation, thereby preserving more information. Fig. B.1 in appendix shows visual examples of where the proposed representation may be helping.

Representation. Conceptually, we perform ray-casting from multiple concentric spheres towards the mesh center, where each sphere has been scaled to different radii. Recording 1st hits from each sphere results in a multi-radius rojection, where we use a uniform $[\frac{1}{R}, \frac{2}{R}, ..., 1]$ radii division assuming normalized inputs. We record distance (with respect to outermost sphere), *sin*, and *cos* features from each ray intersection similar to in other related work (but do not use convex hull information), resulting in 3 features per vertex.

Method	F_1	Params
Cohen et. al. [3] (equiangular, $b = 64$)	0.789[3]	400K[3]
Esteves et. al. [5] (equiangular, $b = 64$)	0.794[3]	500K[3]
DeepSphere [4] (equiangular, $b = 64$)	0.794[3]	190K[3]
DeepSphere [4] (HEALPix, $N_{side} = 32$)	0.807[3]	190K[3]
CSGNN (icosahedral, $L = 4, R = 1$)	0.805	1.3M
CSGNN (icosahedral, $L = 4, R = 16$)	0.823	2.9M

Table 5.2: SHREC17 classification performance in terms of F_1 metric (micro-average). CSGNN is our implementation. Equiangular, HEALPix, and icosahedral are different discretizations of the sphere. CSGNN (this work) uses level 4 icosahedral discretization, R is number of concentric spheres (specific to this work). "Params" is total model size.

Architecture and Hyperparameters. The architecture for SHREC17 is identical to the one used for ModelNet40 in Fig. 5.1, with the exception that the input dimension is 3 (corresponding to features obtained from ray-casting). We consider two model variations, single-sphere ($R = 1$) and multi-sphere ($R = 16$). For $R = 16$, we use a 1D convolution

[3]Numbers as reported in [4].

kernel of size 3. For $R = 1$, we use a 1D convolution kernel of size 1. Note this is equivalent to applying FC layers; we found adding additional FC layers after graph convolutions helped improve performance in the single-sphere case. Training settings are also same as for ModelNet40, except learning rate decay is at epochs 10 and 20. For the $R = 16$ model, each training epoch took 19 minutes, while inference (of validation set) took approximately 2 minutes. Data loading and transformation times are included.

Results. See Table 5.2 for classification results. The reported metric is F_1 micro-average classification score. Results from three other Spherical CNN works designed for general rotational equivariance are included for reference. DeepSphere is most similiar to our work in terms of graph-based spherical convolutions, where the authors explored design of rotationally-equivariant graph convolutional filters with respect to the type of grid and neighborhood size. This work focuses graph construction to the icosahedral discretization, using a minimal set of roughly equidistant neighbors.

We additionally introduce inter-sphere convolutions with concentric spheres, which is largely orthogonal to the design of intra-sphere representation and convolutions. Single-sphere ($R = 1$) CSGNN, a special case of this this work, seems to achieve competitive performance with other single-sphere baselines. However, it is difficult to draw comparative conclusions in this particular case, due to differences in feature extraction, spherical discretization type and size, and model size. More significantly, using multiple spheres ($R = 16$) achieves 2.2% relative performance improvement over the $R = 1$ version. It also seems likely that the concentric spheres approach can be adapted to other spherical convolutional designs as well, but this is beyond the scope of this work.

5.3. Ablation Study. To study the impact of multi-radius spherical discretization, we vary the number of radial levels and present results in Table 5.3. ModelNet40 is used for all ablation experiments. We also use a base model with $R = 16$ and $L = 4$, and keep the number of parameters identical across all cases. For this particular ablation, we use a single channel, indicator feature map–a special case of the RBF mapping where $\gamma = 0$ and $F = 1$. This eliminates γ as a tuning parameter when varying R. Adding radial convolutions in the case of $R = 1$ is equivalent to adding additional dense layers after graph convolutions. We use the same architecture in 5.1, except input dimension is 1. Performance consistently improves with higher radial dimension, peaking at $R = 16$ with 4.8% relative accuracy improvement over the $R = 1$ version. Performance declines for $R = 32$, which suggests diminishing returns (as model capacity may need to be increased).

Setting	$R = 1$	$R = 4$	$R = 8$	$R = 16$	$R = 32$
SO3/SO3	0.839	0.857	0.869	0.879	0.872

Table 5.3: ModelNet40 ablation with number of radial levels (R). Total number of parameters is fixed across all settings.

More ablation studies are presented in Table 5.4. We study the impact of varying the radial kernel size $K_{RC} = [1, 3, 5]$. $K_{RC} = 1$ is same as learning representations independently learned at each radial level. While this still improves over single-sphere representation, using spatial filters ($K_{RC} = [3, 5]$) over the radial dimension is important for best performance. Varying the number of graph and radial convolutional layers per block shows using between 1 and 2 layers per block led to comparable performance. Finally, we compare using either graph convolutions or radial convolutions. Results suggest that it is essential to combine both types of convolutions for best performance. Interestingly, restricting to radial convolutions achieves slightly better performance than restricting to graph convolutions over the

single sphere. This provides further empirical support for the expressiveness of our proposed representation and radial convolutions.

Setting	SO3/SO3
Radial kernel size	
$K_{RC} = 1$, $M_{GC} = 1, M_{RC} = 1$	0.853
$K_{RC} = 3$, $M_{GC} = 1, M_{RC} = 1$	0.880
$K_{RC} = 5$, $M_{GC} = 1, M_{RC} = 1$	0.882
Convolution layers	
$K_{RC} = 3$, $M_{GC} = 1, M_{RC} = 1$	0.880
$K_{RC} = 3$, $M_{GC} = 2, M_{RC} = 2$	0.876
Graph convolution only	
$R = 1$, $M_{GC} = 1$	0.837
Radial convolution only	
$K_{RC} = 3$, $M_{RC} = 1$	0.845

Table 5.4: Ablation study on ModelNet40. K_{RC} is size of radial convolutional kernel, M_{GC} and M_{RC} are number of graph and radial convolutional layers per block. $R = 16$ and $L = 4$, unless stated otherwise.

6. Discussion and Conclusions. In this work we proposed a new multi-sphere convolutional architecture, CSGNN, for learning rotationally invariant representations of 3D data. We introduced distinct intra-sphere and inter-sphere convolutions, which can be combined to learn more expressive representations compared to being restricted to single-sphere representation. Our use of graph and 1D convolutions preserves rotational equivariance, while achieving linear scalability with respect to size of discretization. We achieve state-of-the-art performance in ModelNet classification for testing on arbitrary rotations among both spherical CNN and pointwise convolutional models. We also show that our approach generalizes to classification of 3D mesh obejcts, by improving on single-sphere representations for the SHREC17 task.

One avenue of future work is to explore more descriptive mappings of point cloud data to the discretization. A learned assignment may better learn vertex features for describing nearby points, for instance. There is also room to explore other kinds of convolutions for incorporating inter-sphere information, as well as other radial division schemes. Finally, existing implementations in this work can be more efficient implementations based on using regular properties of the icosahedral grid, as opposed to using a general graph construction.

REFERENCES

[1] C. Chen, G. Li, R. Xu, T. Chen, M. Wang, and L. Lin, *Clusternet: Deep hierarchical cluster network with rigorously rotation-invariant representation for point cloud analysis*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 4994–5002.

[2] T. Cohen, M. Weiler, B. Kicanaoglu, and M. Welling, *Gauge equivariant convolutional networks and the icosahedral CNN*, in International Conference on Machine Learning, 2019, pp. 1321–1330.

[3] T. S. Cohen, M. Geiger, J. Köhler, and M. Welling, *Spherical CNNs*, International Conference on Learning Representations, (2018).

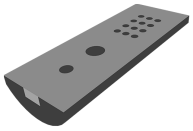
[4] M. Defferrard, M. Milani, F. Gusset, and N. Perraudin, *Deepsphere: a graph-based spherical cnn*, in International Conference on Learning Representations, 2020.

[5] C. Esteves, C. Allen-Blanchette, A. Makadia, and K. Daniilidis, *Learning SO(3) equivariant representations with spherical CNNs*, in Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 52–68.

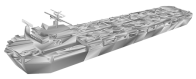
[6] C. M. Jiang, J. Huang, K. Kashinath, Prabhat, P. Marcus, and M. Niessner, *Spherical CNNs on unstructured grids*, in International Conference on Learning Representations, 2019.
[7] T. N. Kipf and M. Welling, *Semi-supervised classification with graph convolutional networks*, International Conference on Learning Representations, (2017).
[8] H.-Y. Meng, L. Gao, Y.-K. Lai, and D. Manocha, *Vv-net: Voxel vae net with group convolutions for point cloud segmentation*, in Proceedings of the IEEE International Conference on Computer Vision, 2019, pp. 8500–8508.
[9] A. Poulenard, M.-J. Rakotosaona, Y. Ponty, and M. Ovsjanikov, *Effective rotation-invariant point cnn with spherical harmonics kernels*, in 2019 International Conference on 3D Vision (3DV), IEEE, 2019, pp. 47–56.
[10] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, *Pointnet: Deep learning on point sets for 3d classification and segmentation*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 652–660.
[11] Y. Rao, J. Lu, and J. Zhou, *Spherical fractal convolutional neural networks for point cloud recognition*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 452–460.
[12] M. Satoh, H. Tomita, H. Yashiro, H. Miura, C. Kodama, T. Seiki, A. T. Noda, Y. Yamada, D. Goto, M. Sawada, et al., *The non-hydrostatic icosahedral atmospheric model: Description and development*, Progress in Earth and Planetary Science, 1 (2014), p. 18.
[13] X. Sun, Z. Lian, and J. Xiao, *Srinet: Learning strictly rotation-invariant representations for point cloud classification and segmentation*, in Proceedings of the 27th ACM International Conference on Multimedia, 2019, pp. 980–988.
[14] N. Thomas, T. Smidt, S. Kearnes, L. Yang, L. Li, K. Kohlhoff, and P. Riley, *Tensor field networks: Rotation-and translation-equivariant neural networks for 3d point clouds*, arXiv preprint arXiv:1802.08219, (2018).
[15] N. Wang and J.-L. Lee, *Geometric properties of the icosahedral-hexagonal grid on the two-sphere*, SIAM Journal on Scientific Computing, 33 (2011), pp. 2536–2559.
[16] Y. You, Y. Lou, Q. Liu, Y.-W. Tai, W. Wang, L. Ma, and C. Lu, *Pointwise rotation-invariant network with adaptive sampling and 3D spherical voxel convolution*, The AAAI Conference on Artificial Intelligence, (2020).
[17] Z. Zhang, B.-S. Hua, D. W. Rosen, and S.-K. Yeung, *Rotation invariant convolutions for 3d point clouds deep learning*, in 2019 International Conference on 3D Vision (3DV), IEEE, 2019, pp. 204–213.

Appendix A. Point Cloud to Spherical Signal. Instead of for each vertex summing over RBF values with respect to all points, for each point we update the features of vertices in its local neighborhood. This is more efficient when the number of points is less than the number of vertices. Ignoring boundary conditions and degenerate cases, any given point p is contained within two bounding "triangles" of the discretization, corresponding to vertices $(u^{(i)}, v^{(i)}, w^{(i)})$ and $(u^{(i+1)}, v^{(i+1)}, w^{(i+1)})$, where i indicates radial level. We define this as the local neighborhood of p. In this case, we can compute Eq. 4.1 with respect to each neighboring vertex and update accordingly. However, using a single γ value for the Gaussian RBF would result in scaling inconsistency: distances between vertices progressively shrink as we move towards inner spheres. Based on the assumption that RBF values should be invariant to scale, we determine a different γ and corresponding RBF depending on radial level. This is determined by first computing the maximum distance d_{max} between any 6 vertices of the two bounding triangles, and using d_{max} to select γ. Specifically, we pick γ such that $\gamma = \frac{-\log T}{d_{max}^2}$, where T is a lower bound target RBF value (tuning parameter). For example, $T = 1$ would correspond to $\gamma = 0$, or a RBF value of 1 for any distance. Note that distances between the bounding vertices $(u^{(i)}, v^{(i)}, w^{(i)})$ and $(u^{(i+1)}, v^{(i+1)}, w^{(i+1)})$ are approximately equal across the same radius level, and so we consistently use one as reference for computing γ.

Appendix B. SHREC17 Visualization. SHREC17 mis-predicted class pairs from single-sphere model are shown in Figure B.1.

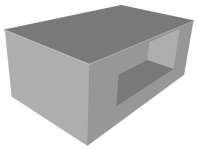


(a) Remote



(b) Watercraft

(c) Cabinet



(d) Table

(e) Lamp

(f) Tower

(g) Dish washer

(h) File cabinet

Fig. B.1: SHREC17 mis-predicted class pairs from single-sphere model, where multi-sphere model showed biggest relative improvement. Each image is a representative sample from the class. Note that *watercraft*, *table*, and *tower* all have more non-convex features that distinguish them from their mis-predicted counterparts. The concentric spherical model seems to better capture these differences.

PERFORMANCE-PORTABLE GRAPH COARSENING METHODS WITH FINE-GRAINED PARALLELISM

MICHAEL S. GILBERT*, KAMESH MADDURI†, SEHER ACER‡, AND SIVASANKARAN RAJAMANICKAM§

Abstract. Graph coarsening methods are an important component of high-performance graph partitioners and PDE solvers. We focus on their impact on graph partitioning, and evaluate the performance of four coarsening methods using two metrics: the time to generate a full hierarchy of coarse graphs, and bipartitioning cutsize resulting from spectral and FM refinement methods. Using a set of large graphs from the SuiteSparse repository, we analyze performance trends using these metrics and compare various coarsening methods. Additionally, we present fine-grained parallel implementations of each coarsening method, and evaluate scaling results and GPU performance.

1. Introduction. Multilevel methods have been an indispensable tool in the task of graph partitioning since their first applications to the problem in the 1990s. Hendrickson et al. [10] and Karypis et al. [13] demonstrated that this methodology is applicable to a large variety of different graph types. The concept is simple: first, approximate an input graph using a hierarchy of monotonically smaller graphs. Second, compute a solution (either a partition or an eigenvector of the graph if using spectral methods) on the smallest graph. Third, project the solution backwards across the hierarchy, performing refinement as needed. See Figure 1.1 for an illustration of this methodology.

Each of these three phases can be done in a variety of ways, and each presents its own challenges when attempting fine-grained parallelism. Bell et al. [2] demonstrate an end-to-end fine-grained parallel method for a multilevel partitioner. We intend to focus on just the first step, graph coarsening, as it is broadly applicable to graph partitioners such as SPHYNX [1] and several available in Zoltan2 [3], multigrid methods such as MueLu [11], and domain decompisition methods such as FROSch [9] (utilizes two levels). We want to answer the following two questions in the context of fine-grained parallel implementations on GPUs: which coarsening method produces a hierarchy of coarse graphs quickly? Which coarsening method produces the highest-quality hierarchy of coarse graphs?

We have built our implementation using the Kokkos [7] library, which enables developers to write applications once and to compile separately for GPUs or CPUs. It provides access to parallel primitives including reductions, mappings, and scans. Additionally, it enables hierarchical parallelism, essentially parallel primitives embedded within other parallel primitives, which can increase the amount of parallelism in a program. This is especially important for GPUs. Kokkos also makes addressing heterogeneous memory simple with its "views", which are managed-memory arrays. Each view automatically resides on the default device (chosen at compile time).

Our contributions are as follows:

- Fine-grained parallel algorithms for the HEC [16], HEM [13], and MT-Metis [15] coarsening heuristics,
- A fine-grained parallel algorithm for coarse graph construction,
- GPU performance comparison of three aforementioned heuristics plus MIS-2 [2],

*Pennsylvania State University, msg5334@psu.edu
†Pennsylvania State University, madduri@psu.edu
‡Sandia National Laboratories, sacer@sandia.gov
§Sandia National Laboratories, srajama@sandia.gov

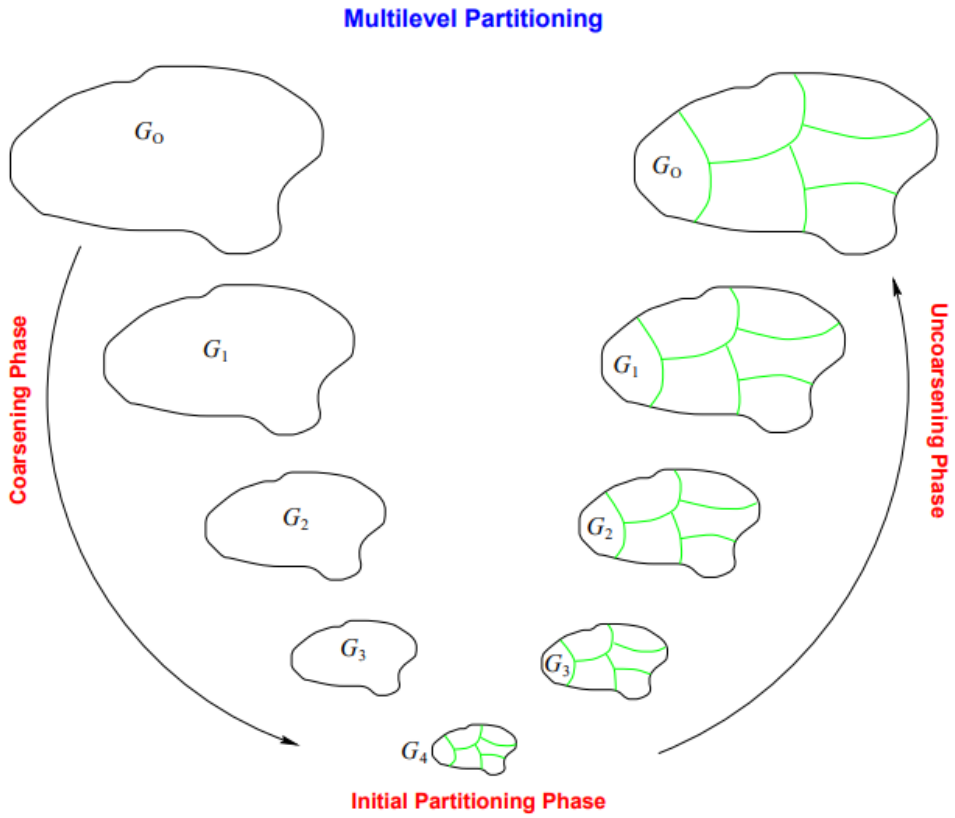


Fig. 1.1: An illustration of multilevel partitioning from the Metis User Manual [12]

- Cutsize comparison of four coarsening heuristics using both FM refinement and spectral partitioning,
- A comparison of cut quality between FM refinement and spectral partitioning,
- A full integration of each GPU coarsening scheme with both FM refinement and spectral partitioning, and
- A full end-to-end graph partitioner on a GPU.

2. Related Work. The analysis of coarsening methods in [4] divides coarsening schemes into two classes: strict aggregation and weighted aggregation. In strict aggregation schemes, each fine vertex belongs entirely to a single coarse vertex, whereas in weighted aggregation, each fine vertex may belong to multiple coarse vertices by fractional weights that sum to 1. In this work, we are solely considering strict aggregation schemes, as our preliminary experiments found that weighted aggregation schemes led to explosive growth in the number of non-zeros between each coarse level. Certain implementations of weighted aggregation schemes such as the one presented in [14] make efforts to limit the total number of coarse vertices a fine vertex can belong to, to preserve sparsity. Regardless, we found that our preliminary implementation of such methods could not scale to the large graphs in our test set.

A variety of strict aggregation coarsening methods exist, and these can be further divided by their ability to avoid "stall" conditions. We use the definition of stall given by Davis et al. [5]. A stall occurs when the number of coarsening iterations required to reduce the vertex count below some threshold becomes proportional to the maximum degree of vertices in the graph. Certain graphs are prone to cause stalling, especially power-law graphs that contain very high-degree vertices. Under practical time limitations, stalls introduce hard constraints on how much these graphs can coarsen. Stalls create a situation where the smallest (coarsest) graph is larger than the chosen threshold, potentially by several orders of magnitude. Davis et al. [5] show results that illustrate a clear cutsize benefit for their stall-free coarsening, and there is reason to believe that this is due to an increased difficulty in finding good initial partitions on these larger graphs.

One of the most important coarsening methods is vertex matching. In such matchings, two vertices may only be matched if they are connected by an edge, and many implementations use maximal matchings [13, 15, 5]. As there is no general lower bound for how small a maximal matching can be, this method is not stall-free. The creators of Metis [13], a popular graph partitioning library, considered four heuristics for creating maximal matchings: random matchings, heavy-edge matchings (HEM), light-edge matchings, and heavy-clique matchings. They found that each method produced similar partition quality, but found that HEM consistently required the least total time between coarsening and uncoarsening phases. The HEM implementation by Karypis and Kumar [13] is serial, and LaSalle et al. [15] give a multicore implementation. In this work we demonstrate a fully fine-grained parallel implementation applicable to GPUs.

The aforementationed work by LaSalle et al. [15] also improved upon HEM to create a stall-free variation. They considered two-hop matches, which are matches between two vertices that are connected by an intermediary vertex. Among two-hop matches, they specify three sub-classes: leaves, twins, and relatives. This augmented version of HEM first performs HEM, and if the ratio of unmatched vertices to total vertices is greater than some threshold, it then performs leaf matches, then twins, and then relatives. If the ratio falls below the threshold at any point, the coarsening iteration finishes. The implementation by LaSalle et al. targets multicore systems, and in this work we target GPUs.

Another coarsening method similar to HEM that shows promise is detailed by Urschel et al. [16], which we denote as HEC. This method removes the strict matching criterion and allows aggregates to form from more than two vertices during each iteration. Instead of matching, each vertex chooses a neighbor adjacent on its heaviest edge to aggregate with, and any number of vertices may participate in each aggregate. Just as a maximal matching has no lower bound in size, so too does the quantity of aggregates produced by HEC. This means that the coarsening may happen arbitrarily quickly, but each coarsening iteration always cuts the number of vertices by at least half. The implementation by Urschel et al. [16] is serial, so we improve upon this with a fine-grained parallel implementation.

The final method we evaluate is the distance-2 maximal independent set (MIS-2) that Bell et al. [2] implemented successfully with fine-grained parallelism. The coarse aggregates are chosen as a subset of the vertices of the fine graph, such that no two aggregates are within a distance of 2 of each other. Each fine vertex is assigned to the nearest aggregate, with ties broken arbitrarily. This is yet another method to achieve stall-free coarsening, and is also the most aggressive. Additionally, it does not consider the edge weights, which contain substantial information about the graph structure, especially in the coarser graphs. We

choose to evaluate this method to compare to our own fine-grained parallel algorithms, as the existing implementation is already fine-grained parallel.

3. Fine-grained Parallel Algorithms for Coarsening Methods. We illustrate our implementation details, with attention to where parallelism is possible.

3.1. Heavy Edge Coarsening (HEC). The HEC algorithm [16] consists of two steps. First, for a graph $G(V, E)$, HEC computes the set of edges $H \subset E$, which contains the heaviest edge adjacent to each vertex. Second, the algorithm traverses the edges in H in a random order, and contracts each edge (u, v) if the source vertex u has not previously participated in a contraction. We generate H using fine-grained parallelism by parallelizing over the source vertices. The edges belonging to each source vertex can be processed in serial, or a parallel argmax reduction may be performed for extra parallelism. We find it convenient to represent H as a many-to-one mapping vector, with each index representing the u and each value representing the v. Fine-grained parallelism in the task of traversing H and performing the contractions is not trivial. To perform a contraction of an edge (u, v), we must ensure that u has never previously participated in a contraction. We could do this with locks, but this did not scale well in our experiments. We show an alternative in Algorithm 1. An execution unit must acquire ownership of both u and v, upon which it generates a new coarse vertex and assigns it to both u and v. If an execution unit can acquire ownership of u but not v, it checks if v has already been assigned a coarse vertex. If this is true, u inherits the coarse vertex of v, otherwise the execution unit releases ownership of u. We parallelize over the source vertices, so any attempt to acquire ownership of u may only fail if some other execution unit has already assigned a coarse vertex to u.

In a cleanup pass, we check which vertices have not been assigned a coarse vertex, and mark them for reprocessing in the next iteration. The algorithm is done when no vertices are marked for reprocessing. It is possible for cyclical ownership conflicts to occur among the edges in H (all edges in the cycle must have the same weight), so we impose an ordering (not shown in algorithm 1) on the vertices that determines which edges in H may be processed in a given iteration. We find that this algorithm requires at $O(iter \times |V|)$ time. Only unaggregated vertices are checked in every pass, but in the worst case where we aggregate only one vertex in a pass, the number of passes needed could be $|V|$.

3.2. Heavy Edge Matching (Matching). Heavy edge matching shares many of the same implementation details as HEC, as matches occur on the same $H \subset E$. However, if some $(u, v) \in H$ can't be matched because v is already matched, a secondary $H_1 \subset E \setminus H$ is used. In practice we do not explicitly create a new H_1, and instead modify the existing H. The need to recompute entries of H is the primary difference in implementation between HEC and matching. This leads to Algorithm 2, given a starting heaviest-edge neighbor vector H:

Algorithm 1 Lock-free Parallel HEC construction

Require: Undirected and connected $G(V, E, W)$. $n = |V|$.
Ensure: $M[1..n]$, n_c.

1: $P[1..n] \leftarrow$ ParGenPerm(n) ▷ parallel sort-based
2: $H[1..n] \leftarrow 0$
3: **for** $u \leftarrow 1$ to n **in parallel do**
4: $w \leftarrow 0$
5: **for** each v adjacent to u **do**
6: **if** $W(u, v) > w$ **then**
7: $w \leftarrow W(u, v)$
8: $H[u] \leftarrow v$
9: **end if**
10: **end for**
11: **end for**
12: $C[1..n] \leftarrow 0$, $M[1..n] \leftarrow 0$, $Q \leftarrow P$, $n_c \leftarrow 1$
13: **for** $i \leftarrow 1$ to $|Q|$ **in parallel do**
14: $u \leftarrow Q[i]$, $v \leftarrow H[u]$
15: **if** $C[u] = 0$ **then**
16: **if** AtomicCAS$(C[u], 0, v) = 0$ **then**
17: **if** AtomicCAS$(C[v], 0, u) = 0$ **then**
18: $m \leftarrow$ AtomicIncr(n_c)
19: $M[u] \leftarrow m$, $M[v] \leftarrow m$
20: **else**
21: **if** $M[v] \neq 0$ **then**
22: $M[u] \leftarrow M[v]$
23: **else**
24: $C[u] \leftarrow 0$
25: **end if**
26: **end if**
27: **end if**
28: **end if**
29: **end for**
30: $R[1..|Q|] \leftarrow 0$
31: **for** $i \leftarrow 1$ to $|Q|$ **in parallel do**
32: $u \leftarrow Q[i]$
33: **if** $M[u] = 0$ **then**
34: Atomically add u to R
35: **end if**
36: **end for**
37: **if** $|R| > 0$ **then**
38: $Q \leftarrow$ NonZeroEntries(R)
39: **go to** line 10
40: **end if**

Algorithm 2 Lock-free Parallel HEM construction

Require: $G(V, E, W)$. $n = |V|$. $H[1..n]$.
Ensure: $M[1..n]$.

1: (P and H are computed as in Alg. 1.)
2: $M[1..n], O[1..n] \leftarrow 0$
3: **for** $i \leftarrow 1$ to n **in parallel do**
4: $\quad O[P[i]] \leftarrow i$ ▷ O is the inverse of P
5: **end for**
6: $C[1..n] \leftarrow 0$
7: $n_c \leftarrow 1$
8: $U \leftarrow P$
9: swap $\leftarrow 0$
10: **while** $U \neq \emptyset$ **do**
11: $\quad$ **for** $u \in U$ **in parallel do**
12: $\quad\quad v \leftarrow H[u]$
13: $\quad\quad$ **if** $(O[u] < O[v]) \oplus$ swap **then**
14: $\quad\quad\quad$ **if** AtomicCAS$(C[u], 0, v) = 0$ **then**
15: $\quad\quad\quad\quad$ **if** AtomicCAS$(C[v], 0, u) = 0$ **then**
16: $\quad\quad\quad\quad\quad m \leftarrow$ AtomicIncr(n_c)
17: $\quad\quad\quad\quad\quad M[u] \leftarrow m$
18: $\quad\quad\quad\quad\quad M[v] \leftarrow m$
19: $\quad\quad\quad\quad$ **else**
20: $\quad\quad\quad\quad\quad C[u] \leftarrow 0$
21: $\quad\quad\quad\quad$ **end if**
22: $\quad\quad\quad$ **end if**
23: $\quad\quad$ **end if**
24: $\quad$ **end for**
25: $\quad R \leftarrow \emptyset$
26: $\quad$ **for** $u \in U$ **in parallel do**
27: $\quad\quad$ **if** $M[u] = 0$ **then**
28: $\quad\quad\quad v \leftarrow$ HeaviestUnmatchedNeighbor(u)
29: $\quad\quad\quad$ **if** $v \neq 0$ **then**
30: $\quad\quad\quad\quad H[u] \leftarrow v$
31: $\quad\quad\quad\quad R \leftarrow R \cup \{u\}$
32: $\quad\quad\quad$ **end if**
33: $\quad\quad$ **end if**
34: $\quad$ **end for**
35: $\quad U \leftarrow R$
36: $\quad$ swap $\leftarrow 1 \oplus$ swap
37: **end while**

Algorithm 2 differs from Algorithm 1 in two ways. The first difference is the action taken if the execution unit fails to acquire ownership of v. Algorithm 2 simply releases ownership of u, whereas algorithm 1 checks if v has a coarse vertex. The second difference is in how vertices are marked for reprocessing. In addition to having no coarse vertex assignment, each u must have some remaining unmatched neighbor. If this is true, $H[u]$ is reassigned to the heaviest remaining unmatched edge. As this algorithm computes a matching, it is likely that many vertices will remain unmatched. We assign these vertices to singleton coarse vertices.

3.3. MT Metis Optimizations. The MT Metis (abbreviated MT hereafter) optimizations [15] build upon a heavy-edge matching with three types of two-hop matches, for the purpose of reducing coarsening stalls. These three types are leaves, twins and relatives, where:

$$leaves \subset twins \subset relatives. \tag{3.1}$$

A relative is any 2-hop match. Twins are relative matches between vertices with identical adjacency lists. Leaves are twins where both vertices have a singular adjacency. The process for assigning coarse vertices proceeds in five phases. If more than 75% of the vertices have been assigned a coarse vertex after any one of the phases, then the process skips to the final phase. In the first phase is the heavy edge matching, which is conducted as in algorithm 2. Leaves, twins, and relatives correspond to the second, third, and fourth phases respectively. In the fifth phase, we assign the remaining vertices to singleton aggregates. See Algorithm 3 for an overview.

Algorithm 3 MT-Metis Optimization Aggregation

Require: $G(V, E, W)$. $n = |V|$.
Ensure: aggregate mapping $M[1..n]$
1: $M \leftarrow \text{HEM}(G(V, E, W))$
2: **if** AGGREGATED_RATIO$(G(V, E, W), M) < .75$ **then**
3: $\quad M \leftarrow$ LEAVES$(G(V, E, W), M)$
4: **end if**
5: **if** AGGREGATED_RATIO$(G(V, E, W), C) < .75$ **then**
6: $\quad M \leftarrow$ TWINS$(G(V, E, W), M)$
7: **end if**
8: **if** AGGREGATED_RATIO$(G(V, E, W), C) < .75$ **then**
9: $\quad M \leftarrow$ RELATIVES$(G(V, E, W), M)$
10: **end if**
11: $M \leftarrow$ SINGLETONS$(G(V, E, W), M)$

Algorithm 4 Parallel Leaf mapping construction

Require: $G(V, E, W)$. $M[1..n]$. $n = |V|$. n_c.
Ensure: $M[1..n]$. n_c.

```
1: for u ← 1 to n in parallel do
2:     if M[u] ≠ 0 then
3:         lastLeaf ← 0
4:         for each v adjacent to u do
5:             if number of v adjacencies is 1 then
6:                 if lastLeaf ≠ 0 then
7:                     m ← AtomicIncr(n_c)
8:                     M[v] ← m
9:                     M[lastLeaf] ← m
10:                    lastLeaf ← 0
11:                else
12:                    lastLeaf ← v
13:                end if
14:            end if
15:        end for
16:    end if
17: end for
```

Algorithm 5 Parallel Twin mapping construction

Require: $G(V, E, W)$. $M[1..n]$. $n = |V|$. n_c.
Ensure: $M[1..n]$. n_c.

```
1: H[1..n] ← 0
2: for u ← 1 to n in parallel do
3:     if M[u] = 0 then
4:         hash ← 0
5:         for each v adjacent to u do
6:             hash ← hash + HASH-FUNCTION(v)
7:         end for
8:         H[u] ← hash
9:     else
10:        H[u] ← hash
11:    end if
12: end for
13: H_s, V_s ← RADIX-SORT-V-AND-H-BY-H(H, V)
14: for scan from i ← 1 to n in parallel do
15:    if H_s[i] ≠ H_s[i − 1] then
16:        scan ← max(scan, i)
17:    end if
18:    if i > scan and i − scan is odd and H_s[i] ≠ 0 then
19:        m ← AtomicIncr(n_c)
20:        M[V_s[i]] ← m
21:        M[V_s[i − 1]] ← m
22:    end if
23: end for
```

Algorithm 6 Parallel Relative mapping construction

Require: $G(V, E, W)$. $M[1..n]$. $n = |V|$. n_c.
Ensure: $M[1..n]$. n_c.

```
1:  H[1..n] ← 0
2:  continue ← 1
3:  while continue do
4:      for u ← 1 to n in parallel do
5:          if M[u] ≠ 0 then
6:              last ← 0
7:              for each v adjacent to u do
8:                  if M[v] = 0 then
9:                      if last ≠ NULL then
10:                         H[v] ← last
11:                         H[last] ← v
12:                         last ← 0
13:                     else
14:                         last ← v
15:                     end if
16:                 end if
17:             end for
18:         end if
19:     end for
20:     U ←MAPPABLE-VERTICES(H, V)
21:     if |U| = 0 then continue ← 0
22:     end if
23:     C[1..n] ← 0
24:     for u ∈ U in parallel do
25:         v ← H[u]
26:         if O[u] < O[v] then
27:             if AtomicCAS(C[u], 0, v) = 0 then
28:                 if AtomicCAS(C[v], 0, u) = 0 then
29:                     m ← AtomicIncr(n_c)
30:                     M[u] ← m
31:                     M[v] ← m
32:                 else
33:                     C[u] ← 0
34:                 end if
35:             end if
36:         end if
37:     end for
38: end while
```

Algorithm 4 shows an $O(|E|)$-time implementation to perform the leaf matchings, but an $O(|V|)$-time implementation is possible. This alternative method is more complex to implement, as it involves counting the leaves for each matched vertex (this only requires examining up to 1 edge per leaf vertex), and writing them into an additional memory array. The twins are a bit more complicated. We use a hash function to create digests of edge-lists in Algorithm 5, so we can quickly determine twins. Algorithm 6 is similar to Algorithm 2 except the H vector denotes relatives, and there are of course no weights to examine. We determine H by examining the edges of previously matched vertices. Entries of H are also written concurrently by multiple execution units, but all the writes are valid relatives so we do not need additional synchronization beyond an atomic write.

3.4. MIS-2. In our implementation of this coarsening method we have closely followed the parallel algorithm provided in the Appendix of [2].

3.5. Constructing Coarse Graph From Mapping. Each of these four methods produces a mapping vector from each fine vertex in V_0 to a coarse vertex in V_1. This mapping defines an interpolation matrix I with $|V_0|$ rows and $|V_1|$ columns. Each index in the mapping defines the row and the value defines the column of an entry in I, each with weight 1. All other entries are 0. If we represent the fine edges E_0 as an adjacency matrix, we may form the coarse edges E_1 as an adjacency matrix with Equation 3.2.

$$E_1 = I^T E_0 I \tag{3.2}$$

With this we may perform the coarse graph construction as one call to a sparse matrix transpose kernel and two calls to a sparse matrix-matrix multiplication kernel. However, the structure of I leads to optimizations not possible when using a general-purpose sparse matrix-matrix multiplication kernel. We propose Algorithm 7 as our alternative. There are four distinct phases to Algorithm 7. In the first phase, it sums the number of entries in each fine row that map to the same coarse row, and performs a prefix sum to calculate the offsets for the start of each coarse row. In the second phase, the algorithm moves the fine entries into the space designated for each coarse row, and also translates these fine entries to coarse entries. In the third phase, the algorithm combines the weights of duplicate coarse entries. In the fourth phase, the algorithm compresses the rows of the coarse matrix. For the deduplication phase, there are two options to perform the deduplication: use a hashmap within each row, or perform a sort within each row.

On CPU, the hashmap is the best option purely for complexity reasons. On GPU, the algorithm must preallocate space for the hashmaps, and each must have enough storage for the largest row in the matrix. This introduces difficulties for power-law graphs, as this greatly limits the number of hashmaps that can be stored in memory. In turn this limits the degree of parallelism possible. This can be remedied by using two tiers of hashmaps, one which is viable for most rows, another which is viable for the few very large rows. As memory is limited, the number of hashmaps that can be preallocated for either level is often a trade-off for larger graphs. We can avoid this by processing the rows in multiple passes, allowing us to use the full memory space for each tier of hashmaps. This also makes it simpler to use an arbitrary number of tiers.

Another wrinkle on the GPU is the need for parallelism within each row to achieve high device utilization, especially for power-law graphs. Unfortunately, the implementation of hashmaps we used did not support concurrent insertion of duplicate values, so we could not make use of parallelism within each row. We found experimentally that this problem greatly limited the efficiency of our implementation.

Algorithm 7 Parallel coarse graph construction.

Require: $G(V, E, W)$, $n = |V|$, $m = |E|$, $M[1..n]$, n_c.
Ensure: $G(V_c, E_c, W_c)$.

```
1: C'[1..n_c] ← 0, C[1..n_c] ← 0
2: for u ← 1 to n in parallel do
3:     for each v adjacent to u do
4:         if M[u] ≠ M[v] then
5:             AtomicIncr(C'[M[u]])
6:         end if
7:     end for
8: end for
9: for u ← 1 to n in parallel do
10:    for each v adjacent to u do
11:        if M[u] ≠ M[v] then
12:            if (C'[M[u]] < C'[M[v]]) or (C'[M[u] = C'[M[v] and u < v) then
13:                AtomicIncr(C[M[u]])
14:            end if
15:        end if
16:    end for
17: end for
18: R ← ParPrefixSums(C)
19: m' ← R[n_c], C[1..n_c] ← 0
20: F[1..m'] ← 0, X[1..m'] ← 0
21: for u ← 1 to n in parallel do
22:    for each v adjacent to u do
23:        if M[u] ≠ M[v] then
24:            if (C'[M[u]] < C'[M[u]]) or (C'[M[u] = C'[M[v] and u < v) then
25:                l ← FindLoc(R, C, u)
26:                F[l] ← M[v], X[l] ← W(u, v)
27:            end if
28:        end if
29:    end for
30: end for
31: for u ← 1 to n_c in parallel do
32:    F, X, C[u] ← DedupWithWts(F, X, R, C, u)
33: end for
34: E_c, W_c, m_c ← GraphConsWithTrans(F, X, R, C')
```

Sorting within each row is only feasible if the sort enables a high degree of parallelism within the row. We chose bitonic sort as it is highly parallel, and found that this generally outperformed our hashmap implementation on the GPU, but not on the CPU. We decided to implement divergent paths in our implementation that utilize hashmaps on the CPU, and utilize the bitonic sort on the GPU.

4. Refinement Methods. In this section we present the two partitioning methods we will use to evaluate our coarsening methods.

4.1. FM Refinement. FM refinement [8] takes an initial partition on a graph and refines with the goal of decreasing the cutsize and imbalance. It is a globally greedy algorithm, which evaluates every vertex and selects the vertex which most decreases the cutsize.

In the case that no vertex decreases the cutsize, the algorithm chooses the vertex which least increases the cutsize. In a single iteration, each vertex can only move once, and each move is recorded, along with the cutsize and imbalance after the move. After all vertices are moved, the algorithm rewinds to the point with the smallest cutsize. We present Algorithm 8 to illustrate this process more concretely. There are a variety of optimizations possible to quickly determine which vertex should move, and to determine when the algorithm can stop early. This algorithm has some potential for parallelism but the core greedy routine can't be parallelised unless global optimality is relaxed. The implementation of the FM algorithm is not the focus of this paper so we leave parallelism here to future work. As this method of refinement needs an initial partition, we use the Greedy Graph Growing Algorithm (GGGP) [13] to create the partition on the coarsest graph. In the case that our coarse graph was too big to use GGGP, we use spectral partitioning to generate a partition on the coarsest graph.

4.2. Spectral Refinement. Spectral refinement does not produce a partition directly. Instead, it utilizes the graph Laplacian L of the graph G, which is formed as follows:

$$L = D - A \tag{4.1}$$

D is a diagonal matrix, and the diagonal entry of a row i is equal to the ith vertices' degree. A is the adjacency matrix of G. The eigenvectors are given as the solution to the following for some constant λ:

$$Lx = \lambda x \tag{4.2}$$

We want to compute the Fiedler vector, the eigenvector corresponding to the second smallest eigenvalue. Given an approximation x_k of the Fiedler vector, we can use power iteration to improve the approximation:

$$x_{k+1} = \frac{Lx_k}{||Lx_k||} \tag{4.3}$$

This is a sparse-matrix vector multiplication, which may be implemented with fine-grained parallelism. Power iteration ceases when the dot product of successive approximations is sufficiently close to 1:

$$|x_{k+1} \cdot x_k - 1| < \delta \tag{4.4}$$

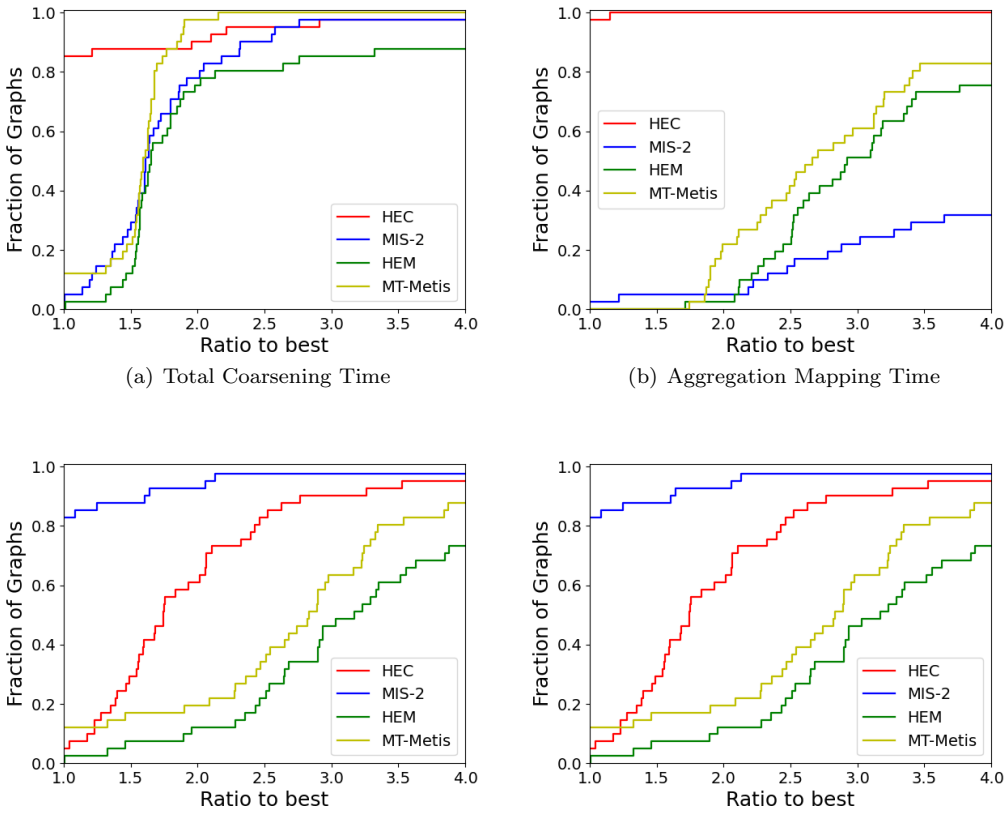
Multi-level methods are important for power iteration because Fiedler vectors of coarse graphs may be projected to the finer graphs to obtain good approximations. This decreases the number of iterations to converge. Given the Fiedler vector, we partition the rows by the median-valued row of the vector. Each vertex in the graph is associated with a row, so vertices with a value lower than the median go in one partition, and the rest go in the other partition.

Algorithm 8 FM Refinement

Require: $G(V, E)$. Partition vector $P_0[1..n]$. $n = |V|$.
Ensure: $P_1[1..n]$

gain$_{max} \leftarrow 0$
cutsize $\leftarrow 0$
balance $\leftarrow 0$
Gains$[1..n] \leftarrow 0$
(Loop over edges to find gains, current total cutsize, and current balance)
for $u = 1$ to n **do**
    $E_u \leftarrow E$ adjacent to u
    Gains$[u] \leftarrow$ COMPUTE-GAIN(u, E_u, P_0)
    cutsize $\leftarrow$ cutsize + COMPUTE-LOCAL-CUT(u, E_u, P_0)
    if $P_0[u] = 0$ **then**
        balance $\leftarrow$ balance + weight(u)
    else
        balance $\leftarrow$ balance - weight(u)
    end if
end for
(Compute maximum possible gain)
for $u = 1$ to n **do**
    $E_u \leftarrow E$ adjacent to u
    sum $\leftarrow$ SUM-EDGE-WEIGHTS(E_u)
    if gain$_{max} <$ sum **then**
        gain$_{max} \leftarrow$ sum
    end if
end for
(One bucket array for each partition, each has 2*maximum gain total buckets)
$B_0, B_1 \leftarrow$ ALLOCATE-BUCKETS$(2 *$ gain$_{max})$
for $u = 1$ to n **do**
    if $P_0[u] == 0$ **then**
        bucket $\leftarrow B_0$[Gains$[u]$]
    else
        bucket $\leftarrow B_1$[Gains$[u]$]
    end if
    INSERT_INTO_BUCKET$(bucket, u)$
end for
swap_sequence $\leftarrow$ empty_list
$P_1 \leftarrow P_0$
while B_0 not empty AND B_1 not empty **do**
    if balance > 0 **then**
        swap $\leftarrow$ BEST-GAIN-IN(B_0)
    else if balance < 0 **then**
        swap $\leftarrow$ BEST-GAIN-IN(B_1)
    else
        swap $\leftarrow$ BEST-GAIN-IN-EITHER(B_0, B_1)
    end if
    (remove swap from datastructure, update cutsize and balance)
    REMOVE-AND-PERFORM-SWAP$(P_1, swap, cutsize, balance)$
    $E_{swap} \leftarrow E_0$ adjacent to swap
    UPDATE-ADJACENT-GAINS$(E_{swap}, swap)$
    APPEND-TO-LIST(swap_sequence,{swap,cutsize,balance})
end while
(Here we find the optimum cutsize and imbalance combination from the list. We then undo all swaps after that point.)
SELECT-BEST-AND-UNDO-REST(swap_sequence, P_1)

5. Results. To evaluate these methods we are using graphs from the SuiteSpare collection [6]. We have chosen to prioritize large graphs with more than 50 million non-zeros, and we chose about 50 graphs of this size that could fit in our GPU memory (11GB NVIDIA RTX 2080 Ti). We ran our Kokkos GPU implementation of each method ten times on each graph, and compare the median timing and cutsize results. For certain combinations of graphs and methods the program either ran out of GPU memory or took longer than our time limit. In these cases we do not include the results from other methods that completed on the graph. We present performance profile graphs as they are the best method to visualize this quantity of results.



(a) Total Coarsening Time

(b) Aggregation Mapping Time

(c) Coarse Graph Construction Time

(d) Total Number of Coarse Levels (hard to see: match runs along x-axis)

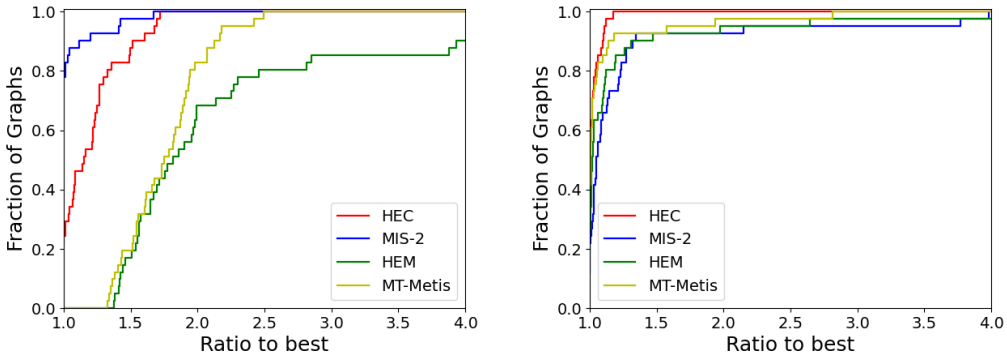
Fig. 5.1: Performance Profiles for Generating Full Coarse Graph Hierarchy

5.1. Coarsening Time Breakdown. In Figures 5.1(a), 5.1(b), 5.1(c), and 5.1(d) we compare each coarsening heuristic on our test graphs. Where applicable, timing information is across the entire hierarchy of coarse graphs. In Figure 5.1(a) we note that HEC is capable of producing the entire coarse graph hierarchy more quickly than the other methods most of the time. Following HEC are MT-Metis and MIS-2 in a tie for second, with HEM in fourth. HEC's success here is due to its aggressiveness, combined with the simplicity of its implementation. MIS-2 is even more aggressive, but as we will note for 5.1(b), its imple-

mentation is not as efficient. MT-Metis and HEM are both far less aggressive, but HEM is prone to stalling so it often has a much deeper hierarchy than any other method.

In Figure 5.1(b) we can see HEC's primary strength, the efficiency of computing the aggregate mapping. HEC and HEM should take approximately the same amount of time on a per level basis since their implementations are very similar (compare Algorithms 1 and 2), but as HEC has far fewer levels (see 5.1(d)), it is much faster than HEM. MT-Metis also benefits from having fewer levels than matching. Although MIS-2 has the shallowest hierarchy overall, the time to compute an aggregate for a single level is the longest of all methods.

Looking at Figures 5.1(c) and 5.1(d) we can see that the coarse graph construction time appears to favor the methods with shallower hierarchies.



(a) FM Refinement Time (includes finding initial partition + partition projection)

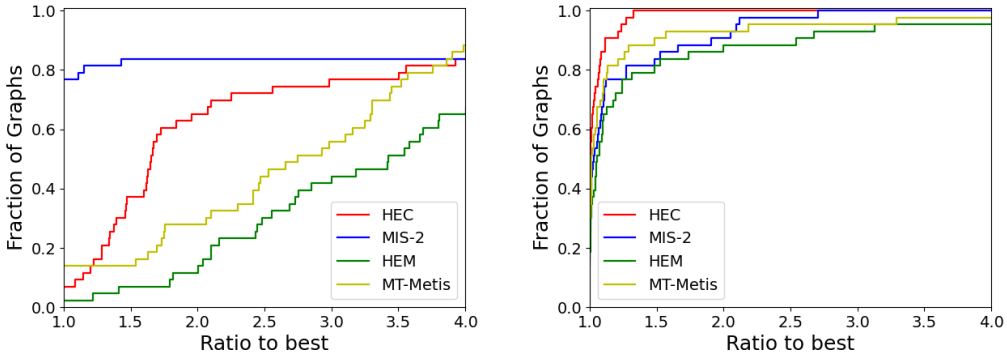
(b) FM Bi-partition Cutsize

Fig. 5.2: FM Refinement Performance Profiles

5.2. FM Refinement Breakdown. In Figures 5.2(a) and 5.2(b) we show refinement times and cutsize results using FM refinement for each of the four coarsening heuristics on our test graphs. The refinement times in Figure 5.2(a) shows a clear trend favoring fewer coarse levels as in Figure 5.1(d). Figure 5.2(b) shows that HEC seems to come out far ahead of the other methods, followed by MT-Metis, then HEM, then MIS-2. We suspect that the weaker performance of MIS-2 is due to it ignoring edge weights, and therefore a substantial amount of structural information.

5.3. Spectral Refinement Breakdown. Figures 5.3(a) and 5.3(b) show the refinement time and cutsize results using spectral partitioning for each of the four coarsening heuristics on our test graphs. The trends in spectral refinement cutsize in Figure 5.3(b) closely mirror those in FM refinement cutsize, with HEC leading the pack. However, we see that MIS-2 pulls ahead of MT-Metis and HEM. The timing trends in Figure 5.3(a) show greater stratification between the best method and the rest of the methods. This is likely due to the eigenvector convergence rate, which is strongly correlated to the quality of coarsenings.

5.4. FM vs Spectral. In our results we additionally looked at the cutsize performance of FM refinement versus spectral refinement. In Figure 5.4 we compare the best cutsize



(a) Spectral Refinement Time (includes finding initial eigenvector guess + eigenvector projection)

(b) Spectral Bi-partition Cutsize

Fig. 5.3: FM Refinement Performance Profiles

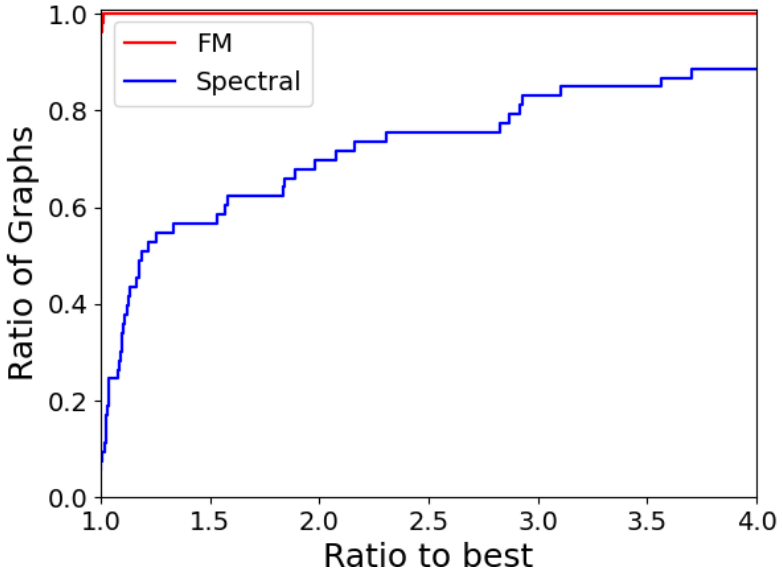


Fig. 5.4: Best Method's Fiduccia-Mattheyses Refinement Bipartitioning Cutsize Vs Best Method's Spectral Bipartitioning Cutsize

achieved by any of the four coarsening methods using FM refinement to the best cutsize achieved by any of the four coarsening methods using spectral refinement. This comparison demonstrates that FM vastly outperforms spectral refinement, on every graph we tested. Furthermore, spectral had 50% larger cutsizes on more than 40% of all graphs. This inspired us to look at the best coarsening method on this subset of graphs where spectral refinement produced 50% worse cutsizes than FM refinement. Comparing 5.5 to 5.2(b), we look at

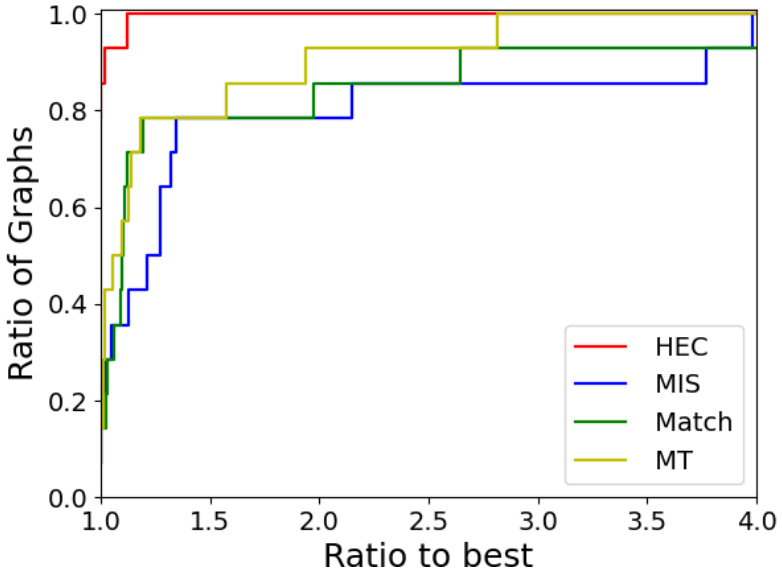


Fig. 5.5: FM cutsize on subset of graphs where best spectral cut at least 50% larger than best FM cut

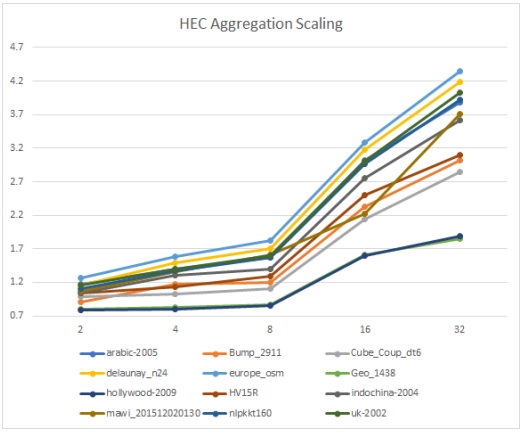
when each method was within 50% of the best bi-partitioning cutsize in Table 5.1. In this table we can see that each method's fraction of graphs decreased, except for HEC. This result indicates HEC's superiority on this subset of graphs, showing that FM refinement is best paired with HEC coarsening.

Method	All Graphs	FM Optimal Subset
HEC	100	100
MT	92	79
MIS	92	79
Match	90	79

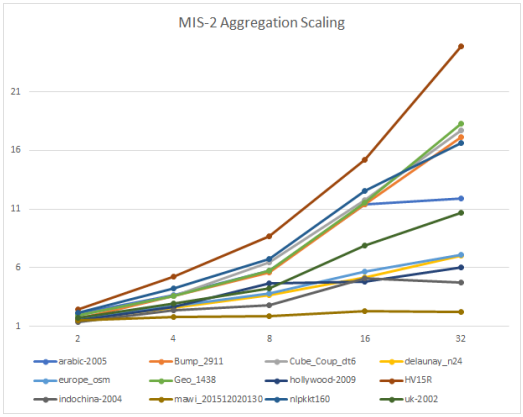
Table 5.1: Percent of Graphs where method was within 50% of best

5.5. Parallel Implementation Scaling Results. In this section we demonstrate scaling results on CPU on a subset of our test graphs, using a 32-core AMD Ryzen Threadripper 3970x with 256GB of RAM. Each datapoint is collected from a single test run, as we observed very little variance in runtimes. An ideal scaling result would show speedup equal to the thread count. In Figures 5.6(a), 5.7(b), 5.7(a), and 5.6(b) we specifically focus on the aggregation methods, which excludes the time spent constructing coarse graphs.

In Figure 5.6(a) we see that our HEC implementation gains the largest speedup going from 8 to 16 threads, with the second largest speedup going from 16 to 32 threads. Overall however, the final speedup at 32 threads is approximately around 3-4x. In Figure 5.6(b) the MIS-2 speedup trend seems to be more regular across each jump in thread count. Additionally, the speedup at 32 threads has a rather large range, and almost all graphs receive

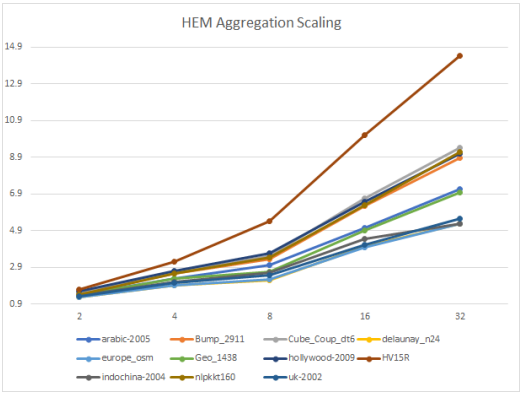


(a) HEC Aggregation Mapping Speedup over 1 Thread

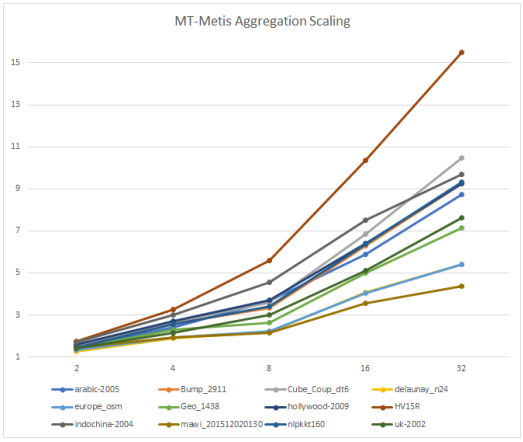


(b) MIS-2 Aggregation Mapping Speedup over 1 Thread

Fig. 5.6: FM Refinement Performance Profiles



(a) HEM Aggregation Mapping Speedup over 1 Thread



(b) MT Aggregation Mapping Speedup over 1 Thread

Fig. 5.7: Aggregation CPU Scaling

a larger total speedup with MIS-2 than HEC. In Figure 5.7(a) we see that HEM has a strong speedup going from 8 to 16 threads, just as with HEC. However, the final speedup for each graph is consistently with HEM than HEC, but slightly less than MIS-2. Since the algorithms to compute the HEC aggregation and HEM aggregation (Algorithms 1 and 2 respectively) are structurally quite similar, we should expect similar trends between HEC and HEM. After some analysis of our implementation, we discovered that we had an atomic counter in both the HEC and HEM implementations. It was the dominant operation in the second loop of algorithm 1, but was surrounded by other expensive operations in the second loop of Algorithm 2. The synchronization time on this atomic became dominant on the HEC implementation but did not in the HEM implementation. Comparing Figures 5.7(b) to 5.7(a), we note that the MT-Metis and HEM aggregation show very similar speedup

trends. We expect this as HEM is a component of the MT-Metis aggregation scheme, and in cases where the HEM algorithm matches 75% of all vertices, the MT-Metis aggregation scheme does no further aggregation.

Out of all methods, HEC has the weakest scaling. We implemented a fix that improved scaling for our CPU testing, but slightly decreased the GPU performance, so we left it as is. Interestingly, three of the four schemes (excluding MIS-2) showed their strongest scaling result going from 8 to 16 threads.

5.6. CPU vs GPU Coarsening Performance. In this section we take a brief look at real timing numbers from the CPU and GPU. We focus on the performance of the HEC coarsening heuristic, as the CPU vs GPU comparison is roughly the same for the other methods. In Figure 5.8 we show the performance of our HEC GPU implementation on a

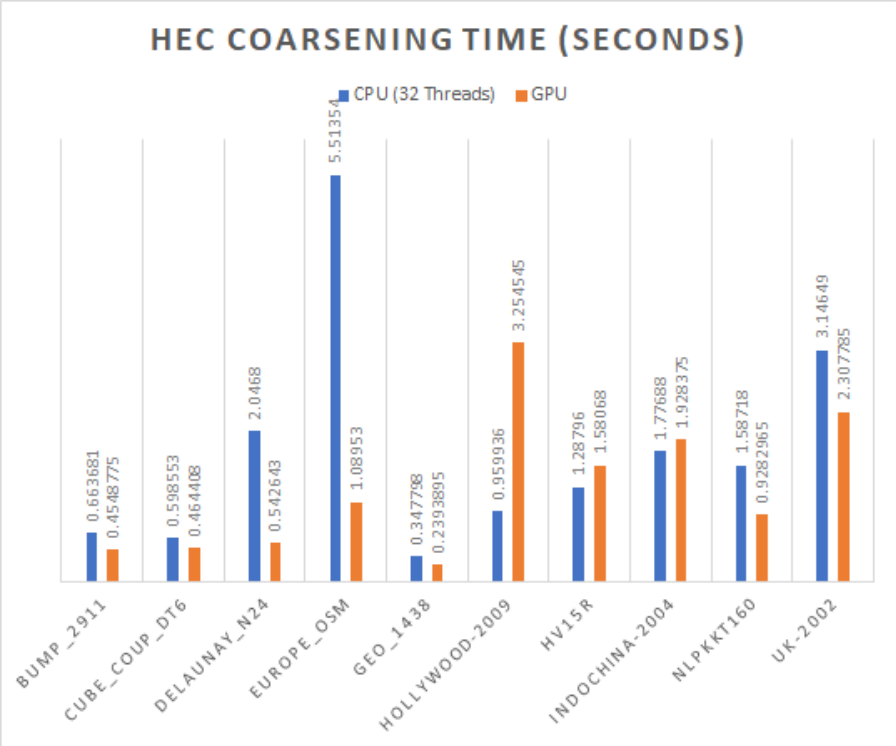


Fig. 5.8: CPU vs GPU Complete Hierarchy Coarsening Time Comparison

small number of graphs. The coarsening performance on GPU compared to CPU shows that our GPU implementation is frequently faster than the CPU, except for a few graphs. Clearly, in Figure 5.9 there is no instance in which the GPU is slower than the CPU, so the cases of slowdown on the GPU are occurring within the construction phase. In further analysis of Hollywood-2009 and Indochina-2004 we found that the max degree of the coarse rows before deduplication was over $1,000,000$ and $200,000$ respectively. We investigated a solution for this problem to reduce this skew, which does not appear in these results. In future work we evaluate this solution, and we find that it is successful in eliminating this slowdown for GPUs. As the GPU uses bitonic sort for deduplication, which has suboptimal

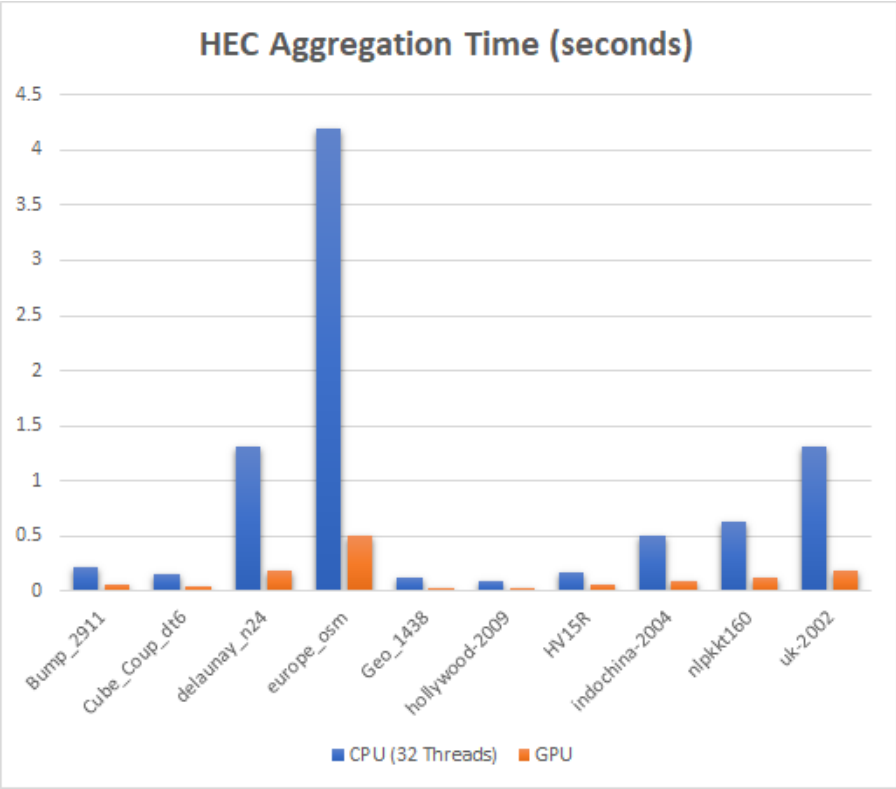


Fig. 5.9: CPU vs GPU Complete Hierarchy Aggregation Time Comparison

time-complexity, this large degree could be the cause for the lack of GPU scaling on these two graphs. We found that HV15R produced a max degree of around $24,000$ in the coarse rows before deduplication, which does not fully explain the lack of scaling here.

6. Conclusions. Considering HEC's dominant performance in terms of cut quality, and very strong performance in terms of time to generate a coarse graph heirarchy on a GPU as well as the time to compute a partition on that heirarchy, HEC is the best choice of coarsening algorithm among those we evaluated. Conversely, HEM should be avoided unless it is known that the graph structure does not exhibit a high risk of stalling. The MT-Metis optimizations and MIS-2 coarsenings are both good options that excel on certain graphs.

In further research, we hope to demonstrate some methods that can reduce the load imbalance of coarse vertex deduplication. We also want to investigate fine-grained parallelism in the refinement stage, specifically parallel analogues to FM refinement. As evidenced in Figure 5.4, FM refinement is a crucial component for a high-quality multi-level partitioner, particularly on irregular graphs. An end-to-end GPU multilevel partitioner that utilizes FM refinement or a relaxation thereof is desirable for this reason. Without this, end-to-end GPU partitioners are restricted to spectral methods. Finally, given the speed of these coarsening methods (particularly HEC), we want to investigate extensions to these methods, particularly using an initial partition of a graph to improve the coarsening quality.

7. Acknowledgements. We would like to thank
- Brian Kelley for writing the GPU implementation of bitonic sort and helping us use GPU hashmaps,
- Erik G Boman for his advice on which coarsening methods to consider, and
- Christian Trott for his help using Kokkos.

REFERENCES

[1] S. Acer, E. G. Boman, and S. Rajamanickam, *Sphynx: Spectral partitioning for hybrid and axelerator-enabled systems*, in 2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2020, pp. 440–449.

[2] N. Bell, S. Dalton, and L. N. Olson, *Exposing fine-grained parallelism in algebraic multigrid methods*, SIAM Journal on Scientific Computing, 34 (2012), pp. C123–C152.

[3] E. G. Boman, K. D. Devine, V. J. Leung, S. Rajamanickam, L. A. Riesen, M. Deveci, and U. Catalyurek, *Zoltan2: Next-generation combinatorial toolkit.*

[4] C. Chevalier and I. Safro, *Comparison of coarsening schemes for multilevel graph partitioning*, in Learning and Intelligent Optimization, T. Stützle, ed., Berlin, Heidelberg, 2009, Springer Berlin Heidelberg, pp. 191–205.

[5] T. A. Davis, W. W. Hager, S. P. Kolodziej, and S. N. Yeralan, *Algorithm 1003: Mongoose, a graph coarsening and partitioning library*, ACM Trans. Math. Softw., 46 (2020).

[6] T. A. Davis and Y. Hu, *The University of Florida sparse matrix collection*, ACM Transactions on Mathematical Software, 38 (2011).

[7] H. C. Edwards, C. R. Trott, and D. Sunderland, *Kokkos: Enabling manycore performance portability through polymorphic memory access patterns*, Journal of Parallel and Distributed Computing, 74 (2014), pp. 3202 – 3216. Domain-Specific Languages and High-Level Frameworks for High-Performance Computing.

[8] C. M. Fiduccia and R. M. Mattheyses, *A linear-time heuristic for improving network partitions*, in 19th Design Automation Conference, 1982, pp. 175–181.

[9] A. Heinlein, A. Klawonn, S. Rajamanickam, and O. Rheinbach, *Frosch: A fast and robust overlapping schwarz domain decomposition preconditioner based on xpetra in trilinos.*

[10] B. Hendrickson and R. Leland, *The chaco users guide. version 1.0*, tech. rep., Sandia National Labs., Albuquerque, NM (United States), 1993.

[11] J. J. Hu, A. Prokopenko, T. A. Wiesner, C. Siefert, and R. S. Tuminaro, *Muelu user's guid for trilinos version 11.12.*

[12] G. Karypis, *METIS: A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices Version 5.1.0*, 2013.

[13] G. Karypis and V. Kumar, *A fast and high quality multilevel scheme for partitioning irregular graphs*, Siam Journal on Scientific Computing, 20 (1998).

[14] Y. Koren, L. Carmel, and D. Harel, *Drawing huge graphs by algebraic multigrid optimization*, Multiscale Modeling & Simulation, 1 (2003), pp. 645–673.

[15] D. LaSalle, M. M. A. Patwary, N. Satish, N. Sundaram, P. Dubey, and G. Karypis, *Improving graph partitioning for modern graphs and architectures*, in Proceedings of the 5th Workshop on Irregular Applications: Architectures and Algorithms, IA3 '15, New York, NY, USA, 2015, Association for Computing Machinery.

[16] J. C. Urschel, J. Xu, X. Hu, and L. T. Zikatanov, *A cascadic multigrid algorithm for computing the fiedler vector of graph laplacians*, Journal of Computational Mathematics, 33 (2015), pp. 209–226.

Appendix A. Graph Listings. See Table A.1 for graphs used in Figures 5.1(a), 5.1(b), 5.1(c), 5.2(a), 5.2(b), and 5.4. See Table A.2 for graphs used in Figures 5.1(d), 5.3(a), and 5.3(b). See Table A.3 for graphs used in Figure 5.5 and Table 5.1.

kron_g500-logn20	ML_Geer
HV15R	bcsstk25
wb-edu	rgg_n_2_24_s0
cage15	nlpkkt120
soc-LiveJournal1	Cube_Coup_dt6
hugebubbles-00020	Geo_1438
rgg_n_2_22_s0	stokes
Cube_Coup_dt0	Queen_4147
Hook_1498	hugebubbles-00000
com-Orkut	ljournal-2008
Long_Coup_dt0	Long_Coup_dt6
circuit5M	audikw_1
mycielskian18	kmer_V2a
af_shell10	channel-500x100x100-b050
Bump_2911	vas_stokes_2M
kmer_U1a	vas_stokes_4M
road_usa	hugebubbles-00010
rgg_n_2_23_s0	nlpkkt160
fe_rotor	indochina-2004
europe_osm	delaunay_n24
hollywood-2009	Flan_1565
mawi_201512020030	com-LiveJournal
dielFilterV3real	kron_g500-logn21
Serena	delaunay_n23
mawi_201512020000	uk-2002
GAP-road	mycielskian17
nlpkkt200	

Table A.1: Graphs used for FM refinement partitioning experiments

kron_g500-logn20	Serena
vas_stokes_2M	HV15R
nlpkkt160	uk-2002
fe_rotor	delaunay_n24
hollywood-2009	Bump_2911
ljournal-2008	delaunay_n23
Long_Coup_dt0	com-LiveJournal
stokes	kron_g500-logn21
audikw_1	hugebubbles-00000
af_shell10	kmer_U1a
road_usa	rgg_n_2_22_s0
kmer_V2a	europe_osm
circuit5M	wb-edu
mycielskian17	bcsstk25
soc-LiveJournal1	mawi_201512020000
com-Orkut	GAP-road
rgg_n_2_23_s0	Long_Coup_dt6
nlpkkt200	Geo_1438
cage15	Cube_Coup_dt6
vas_stokes_4M	indochina-2004
nlpkkt120	Queen_4147
mycielskian18	hugebubbles-00020
mawi_201512020030	hugebubbles-00010
rgg_n_2_24_s0	channel-500x100x100-b050
dielFilterV3real	Cube_Coup_dt0
Flan_1565	ML_Geer
Hook_1498	mawi_201512020130

Table A.2: Graphs used for spectral partitioning experiments

kron_g500-logn20	rgg_n_2_24_s0
cage15	soc-LiveJournal1
rgg_n_2_22_s0	ljournal-2008
kmer_V2a	kmer_U1a
road_usa	rgg_n_2_23_s0
europe_osm	hollywood-2009
com-LiveJournal	GAP-road

Table A.3: Graphs for which FM refinement partitioning produced substantially better cutsizes than spectral partitioning

RADIAL BASIS FUNCTIONS IN THE TANGENT PLANE: MESHFREE APPROXIMATION METHODS FOR MANIFOLDS

ANDREW M. JONES* AND PETER A. BOSLER†

Abstract. This work examines the use of Radial Basis Function (RBF) methods for interpolation and construction of surface differential operators for the unit sphere. We present convergence results for the approximation of the Laplace-Beltrami operator and interpolation of a spherical harmonic. Additionally, we do a comparative analysis of the Generalized Moving Least Squares (GMLS) package Compadre on two different node sets on the unit sphere, equiangular cube sphere and icosahedral nodes. Algorithmic performance analysis is also provided to estimate the cost of using RBF methods.

1. Introduction. Interpolation and approximation of derivatives on surfaces is an important problem that arises in many areas of science and engineering. For example, a challenging problem in atmospheric sciences is the simulation of the shallow wave equations on a rotating sphere [5], which requires both interpolation and approximation of surface differential operators such as the Laplace-Beltrami operator. In this work, we examine the use of RBFs for this task.

RBFs were originally developed in the 1970s for scattered data interpolation problems arising in cartography [7]. Since then they have been extended to many other applications including solving PDES [11, 9, 3, 5]. In this area, RBFs are used for approximating spatial derivatives, for which they can achieve high orders of accuracy. Relevant to this paper we utilize RBFs for approximating surface differential operators. Several studies have shown that this approach works well for problems in atmospheric sciences, such as the shallow water wave equations [3, 5], and in problems in biology, such as pattern forming reaction-diffusion equations [9, 11] . The focus of this study is surface interpolation and approximation of the surface Laplacian of functions on the sphere using the recently developed method called the RBF tangent plane method (RBF-TPM) [12]. The second focus is on the comparisons of RBF-TPM with GMLS [10].

The remainder of the paper can be outlined as follows: In Section 2, we briefly introduce RBFs, describe how they are used in finite difference-type mode, and then describe the tangent plane method (TPM). We then introduce our new software package, RbfKokkos, that uses the Kokkos programming model and its associated libraries to implement RBF-FD. Finishing the work, we present convergence results for RbfKokkos vs. the GMLS package Compadre [8], and a performance analysis of the RBF and GMLS algorithms.

2. Methods.

2.1. RBF Interpolation.

The RBF interpolation process is illustrated in Figure 2.1. This Figure shows a reconstruction of data collected at scattered nodes using Gaussian RBFs. Starting with scattered data, the RBF method produces an interpolant of this data using shifts and rotations of single radial kernel. Gaussians are centered at each of data sites.

A radial kernel $\Phi(\boldsymbol{x}, \boldsymbol{x}_j) := \phi(||\boldsymbol{x} - \boldsymbol{x}_j||)$ is defined by the $||\cdot||$ standard Euclidean norm between distinct point sets, $X = \{\boldsymbol{x}_k\}_{k=1}^{N} \subset \Omega$ in $\mathbb{R}^d$. For RBF interpolation methods one computes distances between a central node $\boldsymbol{x}_j$ and $\boldsymbol{x}$ the evaluation sites for N nodes in Ω.

*Boise State University, andrewjones237@u.boisestate.edu
†Sandia National Laboratories, pabosle@sandia.gov

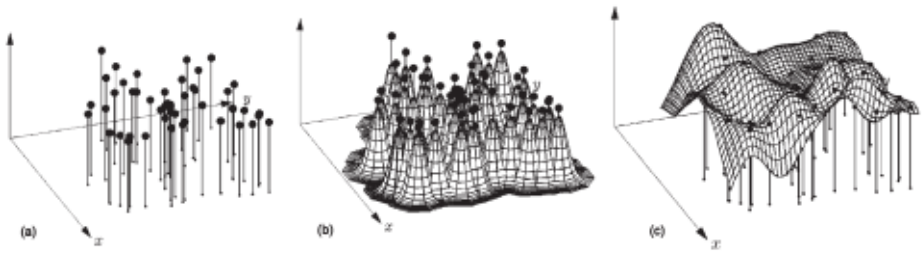


Fig. 2.1: RBF interpolation of 2D scattered node data from [4]. Each subfigure is as follows (a) scattered data (b) radial basis functions (Gaussians), and (c) interpolant of or surface reconstruction (a) .

The basic RBF expansion takes the form,

$$s(\boldsymbol{x}) = \sum_{j=1}^{N} c_j \phi(||\boldsymbol{x} - \boldsymbol{x}_j||). \tag{2.1}$$

The first step of a interpolation problem is finding the coefficients of said expansion, which is found by solving the following system,

$$\sum_{j=1}^{N} c_j \phi(||\boldsymbol{x}_i - \boldsymbol{x}_j||) = f(\boldsymbol{x}_i) \ , \ i = 1, \ldots, N, \tag{2.2}$$

where we have restricted the interpolant to the function at our data sites or input data by $s|_X = f|_X$. This linear system can be written as follows:

$$A\boldsymbol{c} = \boldsymbol{f},$$

where the symmetric matrix A is the global RBF interpolation matrix, $\boldsymbol{c}$ is a vector containing the expansion coefficients, and $\boldsymbol{f}$ is the samples of the target function to be interpolated. The matrix elements are,

$$\begin{bmatrix} \phi(||\boldsymbol{x}_1 - \boldsymbol{x}_1||) & \ldots & \phi(||\boldsymbol{x}_1 - \boldsymbol{x}_N||) \\ \vdots & \ddots & \vdots \\ \phi(\boldsymbol{x}_N - \boldsymbol{x}_1||) & \ldots & \phi(\boldsymbol{x}_N - \boldsymbol{x}_N||) \end{bmatrix} \begin{bmatrix} c_1 \\ \vdots \\ c_N \end{bmatrix} = \begin{bmatrix} f_1 \\ \vdots \\ f_N \end{bmatrix}. \tag{2.3}$$

Some commonly used RBFs are shown in Table 2.1.

Type of RBF	**RBF** $\phi(\boldsymbol{r})$
Polyharmonic Spline (PHS)	$r^{2\ell}\log(r), \quad \ell \in \mathbb{Z}^+ > 0$
Polyharmonic Spline (PHS)	$r^{2\ell+1}, \quad \ell \in \mathbb{Z}^+ \geq 0$
Multiquadric(MQ)	$\sqrt{1+(\epsilon r)^2}$
Gaussian(GA)	$e^{-(\epsilon r)^2}$
Sech(SH)	$\mathrm{sech}(\epsilon r)$

Table 2.1: Commonly used radial basis functions (RBFs).

In this study we use the polyharmonic splines (PHS) which are advantageous because they don't require shape parameters (ϵ) like the Gaussian and Multiquadric RBFs. Choosing good shape parameters often requires expensive optimization algorithms [2], and the interpolation matrix can become extremely ill-conditioned, prompting the use so called stable algorithms [5].

When using PHS, one often appends on low degree polynomials to Eq. (2.1), to guarantee well-posedness of the interpolation problem, but more importantly this has the added benefit of improving approximation convergence rates and allows for polynomial reproduction [5]. The new form of the interpolant is as follows:

$$s(\boldsymbol{x}) = \sum_{j=1}^{N} c_j \phi(||\boldsymbol{x} - \boldsymbol{x}_j||) + \sum_{k=1}^{L} \gamma_k p_k(\boldsymbol{x}). \tag{2.4}$$

The respective terms in the new expansion Eq. (2.4) are: L the dimension of the polynomial space of degree ℓ and $\{p_k\}$ which is a polynomial basis for this space. To account for the new L coefficients, γ_k, the interpolant is subject to the following moment conditions [11]:

$$\sum_{k=1}^{N} c_k p_j(\boldsymbol{x}_k) = 0, \;\; j = 1, 2, \ldots, L.$$

The linear system for determining the expansion coefficients is given as follows:

$$\begin{bmatrix} A & P \\ P^T & O \end{bmatrix} \begin{bmatrix} \boldsymbol{c} \\ \boldsymbol{\gamma} \end{bmatrix} = \begin{bmatrix} \boldsymbol{f} \\ 0 \end{bmatrix}, \tag{2.5}$$

where the block P has entries $P_{jk} = p_k(\boldsymbol{x}_j)$ For example, the elements of the block matrix-vector system are follows for $\ell = 1$,

$$\begin{bmatrix} \Phi(\boldsymbol{x}_1, \boldsymbol{x}_1) & \cdots & \Phi(\boldsymbol{x}_1, \boldsymbol{x}_N) & 1 \\ \vdots & \ddots & \vdots & \vdots \\ \Phi(\boldsymbol{x}_N, \boldsymbol{x}_1) & \cdots & \Phi(\boldsymbol{x}_N, \boldsymbol{x}_N) & 1 \\ 1 & \cdots & 1 & 0 \end{bmatrix} \begin{bmatrix} c_1 \\ \vdots \\ c_N \\ \gamma_1 \end{bmatrix} = \begin{bmatrix} f_1 \\ \vdots \\ f_N \\ 0 \end{bmatrix}. \tag{2.6}$$

One issue with the global interpolation methods presented above is the linear systems 2.3 or 2.5 are dense and not well suited for iterative methods. Direct methods require $O(N^3)$ computational cost, which becomes prohibitive for large N. In the next section we examine using the more computationally efficient local RBF interpolants, and their extension to finding finite difference (FD) weights.

2.2. RBF-FD Direct Method. Local interpolation methods maintain high accuracy, while avoiding constructing and solving of ill-conditioned dense linear systems. The aim of this local method as illustrated in Figure 2.2, is to compute weights for interpolation or approximating derivatives at a target point $\boldsymbol{x}_c$ using stencils consisting of its n nearest neighbors at $\boldsymbol{x}_c$.

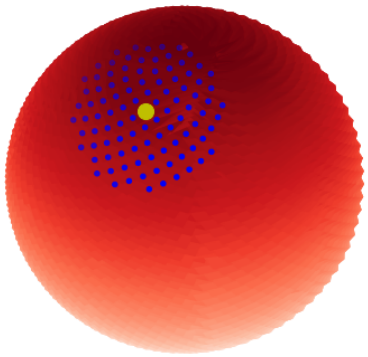


Fig. 2.2: Local interpolation stencil. Nearest neighbors (blue) for a point $\boldsymbol{x}_c$ at a central node (yellow) on a sphere (red).

To determine these weights we use RBFs rather than the traditional choice at polynomials. For a equally spaced grids, the RBF method can be shown to reproduce standard FD stencil weights, however for the RBF-FD method we can maintain high order accuracy on non-uniform meshes or point sets [5, 11, 9].

The RBF-FD weights w_j for the linear differential operator $\mathcal{L}$ are given as a solution to the following system,

$$\sum_{j=1}^{n} w_j \phi(||\boldsymbol{x}_j - \boldsymbol{x}_i||) = \mathcal{L}\phi(||\boldsymbol{x} - \boldsymbol{x}_i||)|_{\boldsymbol{x}=\boldsymbol{x}_c} \ , \ i = 1, \ldots, n, \tag{2.7}$$

where $\boldsymbol{x}_c$ is the stencil's center under the constraint that weights $\boldsymbol{\omega}$ be exactly the finite difference weights for the polynomials of order L for $p_k(\boldsymbol{x})$,

$$\sum_{j=1}^{n} w_k p_k(\boldsymbol{x}_j) = \mathcal{L}p_k(\boldsymbol{x})|_{\boldsymbol{x}=\boldsymbol{x}_c} \quad k = 1, \ldots, L,$$

where $\{p_k\}$ are a basis for the set of polynomials of degree ℓ. As described in detail by [5, pp. 111-112], the weights satisfying Eq. (2.7) under the constraint, can be computed by solving the following linear system,

$$\begin{bmatrix} A & P \\ P^T & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{w} \\ \boldsymbol{\omega} \end{bmatrix} = \begin{bmatrix} \mathcal{L}\phi(||\boldsymbol{x} - \boldsymbol{x}_j||)|_{\boldsymbol{x}=\boldsymbol{x}_c} \\ \mathcal{L}P(\boldsymbol{x})|_{\boldsymbol{x}=\boldsymbol{x}_c} \end{bmatrix}. \tag{2.8}$$

We compute RBF-FD weights according to Eq. (2.8) for each of the N nodes $\boldsymbol{x}_j$ in X. These weights can then be combined into RBF-FD differentiation matrices.

2.3. Tangent Plane Method (TPM) for the Sphere. Surface differential operators for simple geometries (e.g. sphere, torus, ellipsoid, etc.) can be derived explicitly. For more complex geometries these differential operators cannot be easily derived (e.g. bumpy sphere, tooth, frog, Stanford Bunny). Because always a parametric or implicit representations for the $\mathcal{S}$. In this section we explore a simple yet novel method for approximating the Laplace-Beltrami operator $\Delta_{\mathcal{S}}$ on simply connected, closed surfaces that does not require parametric representations.

The Tangent Plane Method (TPM) is first introduced in [1] allows for simple approximation of $\Delta_{\mathcal{S}}$ [1] by projecting the stencil points into the local tangent plane about the

target node $\boldsymbol{x}_c$, as shown by Figure 2.3. After projecting the points, one can just use the standard Laplacian in the plane to approximate $\Delta_{\mathcal{S}}$ at the target $\boldsymbol{x}_c$. To make this procedure easier, we can rotate the projected stencil points to be parallel to the z plane, and just carryout the approximations using the standard Laplacian in $\mathbb{R}^2$. This procedure is carried

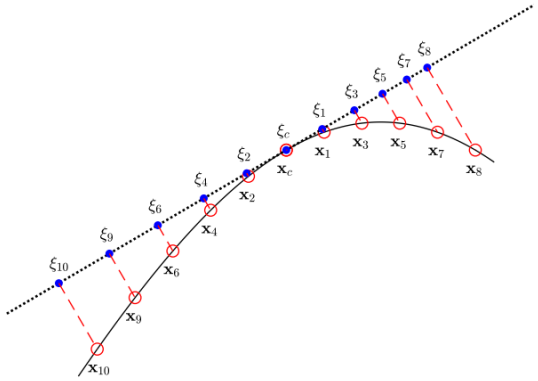


Fig. 2.3: Projection of stencil along stencil centered plane.

out using the projected $W = I - \hat{\boldsymbol{n}}\hat{\boldsymbol{n}}^T$, and $\hat{\boldsymbol{n}}$ is the unit normal vector of the surface $\mathcal{S}$ at $\boldsymbol{x}_c$, and rotation matrix Q,

$$Q = \begin{bmatrix} \hat{\boldsymbol{t}}_1 & \hat{\boldsymbol{t}}_2 & \hat{\boldsymbol{n}} \end{bmatrix},$$

where $\hat{\boldsymbol{t}}_1$ and $\hat{\boldsymbol{t}}_2$ are orthogonal unit tangent vectors spanning the plane orthogonal to $\hat{\boldsymbol{n}}$. The projected and rotated points are as follows:

$$\boldsymbol{\xi}_j = Q^T W \boldsymbol{x}_j.$$

The rotation matrix Q can be found by computing the SVD or any orthogonal matrix decomposition (e.g. QR) of W. When dealing with a surface that has no explicit or implicit parameterization using an orthogonal matrix decomposition is necessary, but for the sphere one could simply compute the meridional and equatorial vectors for a stencil to find the first two columns of Q. The algorithms for implementing the tangent plane interpolation and generation of Laplace-Beltrami weights are shown in the Appendix A.

2.4. Software Design. The RbfKokkos software package uses C++ header based programming style and utilizes the Kokkos library from Sandia National Laboratories for its data structures and parallel efficiency. For the construction and search of the nearest neighbors the open source KDTree package NanoFlann is available also utilization of another open source KDTree package [6]. Currently the code runs the Kokkos based algorithms in serial, and is being adapted to the backend parallelism with Kokkos using OpenMP.

3. Results and Algorithmic Performance.

3.1. Problem Statement. We test the accuracy of the RBF methods for interpolation of the spherical harmonic Y_5^4 which is shown by Figure 3.1,

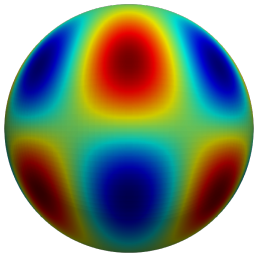


Fig. 3.1: Y_5^4 spherical harmonic on $N = 9128$ cubed sphere nodes.

and we examine the Laplace Beltrami operator applied to a spherical harmonic, which is defined as follows,

$$\Delta_{\mathcal{S}} Y_l^m = -l(l+1) Y_l^m. \tag{3.1}$$

3.2. Convergence Results. The convergence tests are done on icosahedral and equiangular cubed sphere node sets. For the GMLS and RBF software packages, we use for node sets up to $N > 10^6$ and $N > 10^5$ for the surface Laplacian and the interpolation problems. Figures [3.2,3.3], display the convergence results for the interpolation and Laplacian using the ℓ^∞ norm.

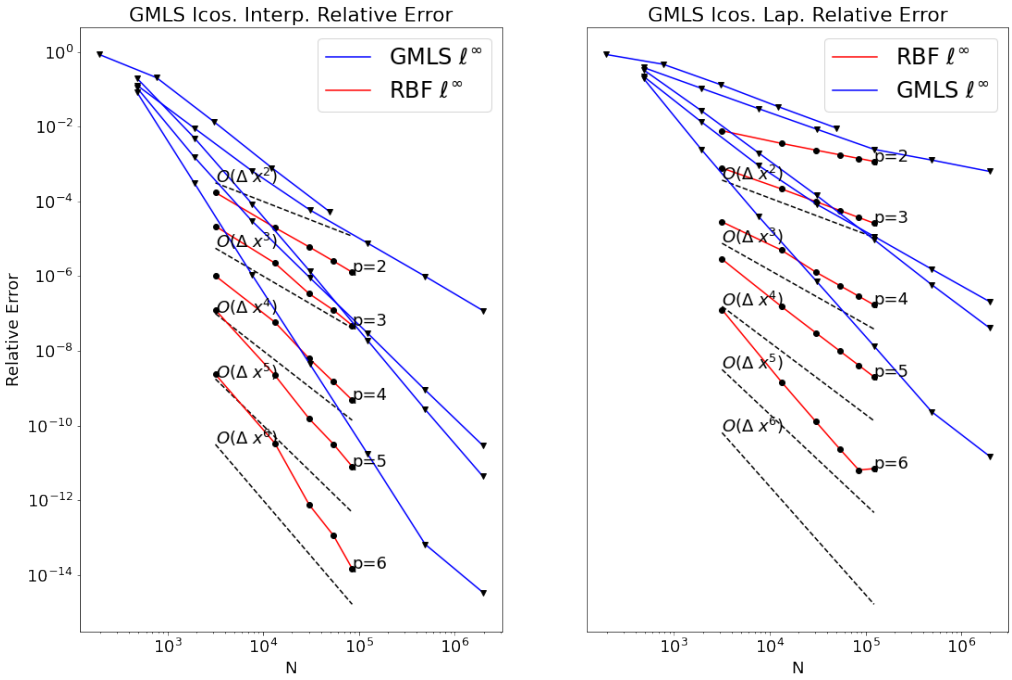


Fig. 3.2: Convergence results for interpolation (left) and $\mathcal{L}_{\mathcal{S}} Y_l^m$ (right) of spherical harmonic on icosahedral nodes. The figures display the decreasing relative ℓ^∞ error as the number of nodes N increase.

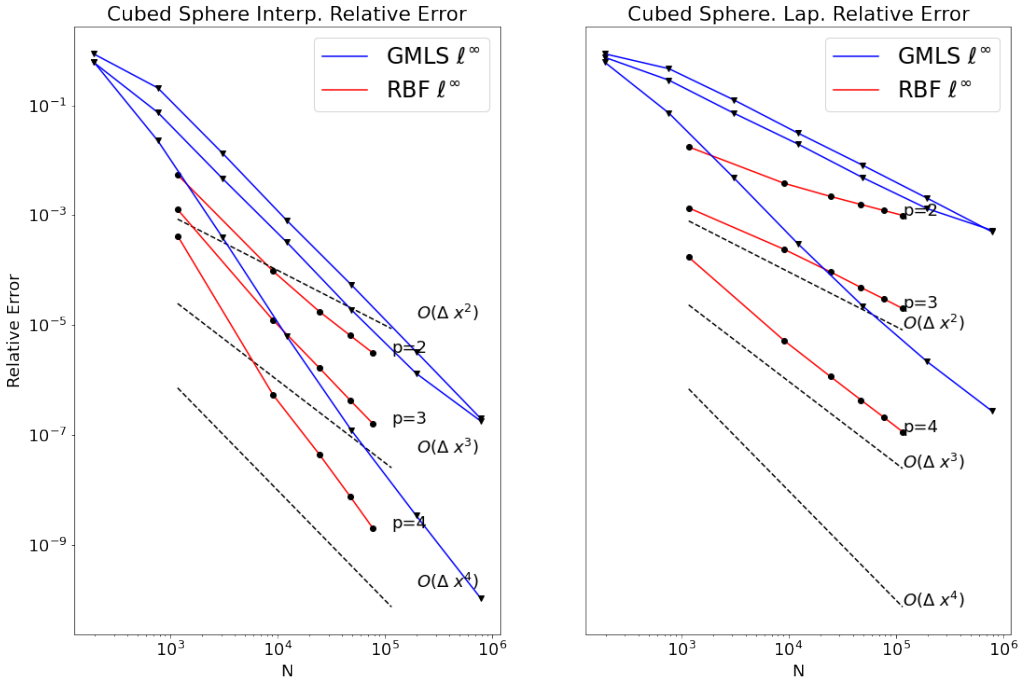


Fig. 3.3: Convergence results for interpolation (left) and $\mathcal{L}_S Y_l^m$ (right) of spherical harmonic on equiangular cubed sphere nodes. The figures display the decreasing relative ℓ^∞ error as increasing number of nodes N.

Which is defined as follows,

$$rel.\ \ell^\infty\ error = \frac{\max_i |(u_i^{approx.} - u_i^{exact})|}{\max_i |u_i^{exact}|}, \tag{3.2}$$

where the exact and the approximate of the solution are, u^{exact} and $u^{approx.}$, respectively.

3.3. Algorithmic Performance. The results in Section 3.2, demonstrate that the RBF methods in RbfKokkos can achieve lower error than the GMLS methods in Compadre. However, we also need to determine which method is more computationally expensive in terms of weight generation and evaluation cost. In this section we examine computational cost of each method. The stencil sizes (n_{rbf}, n_{gmls}) of both methods determine the cost, which for both depends of the dimension of the basis L,

$$L = \frac{(\ell+1)(\ell+2)}{2}, \tag{3.3}$$

where the basis are bivariate polynomials of degree ℓ. RbfKokkos and Compadre both rely on the degree of basis ℓ, but each computes the neighbors differently. RbfKokkos selects the number of nearest neighbors as follows,

$$n_{rbf} = \text{floor}(\alpha L),$$

for this work we select $\alpha = 3.5$, and the n_{rbf} nearest neighbors are found using a KNN search. For Compadre's GMLS method a radius search for finding the nearest neighbors. We define the stencil $\mathcal{S}_0$ mesh width to be,

$$h = \max_{\boldsymbol{x}_i \in \mathcal{S}_0} |\boldsymbol{x}_i - \boldsymbol{x}_c|, i = 1, \ldots, L,$$

where $\boldsymbol{x}_c$ is the stencil center. The search radius is h, scaled by a user-chosen parameter $\tau > 1$. Shown here,

$$\tau = \begin{cases} 2 & , \quad \ell \leq 4, \\ 3 & , \quad \ell > 4. \end{cases}$$

Then n_{gmls} is the number of points in the set $\boldsymbol{x}_i \in X : |\boldsymbol{x}_i - \boldsymbol{x}_c| < \tau h$

According to Eq. (2.5) the RBF system is $(n_{rbf} + L) \times (n_{rbf} + L)$, which is solved takes $O((n_{rbf} + L)^3)$ operations via a Householder QR algorithm. For GMLS one must solve a Least Squares problem, which takes $O(n_{gmls} L^2)$ and also uses a Householder QR algorithm. The weight generation stage for both methods can be carried out before simulation time, so the per time step cost for a simple time dependent PDE of the form,

$$\frac{d\boldsymbol{u}}{dt} = \mathcal{L}_S \boldsymbol{u}, \tag{3.4}$$

using an explicit time-stepping scheme requires $O(sN)$ operations, where s is the bandwidth of $\mathcal{L}_S$. We can make the assumption that $s = n$ for each of the respective methods, since we can compute the inner product of the target function and the weights so our evaluation cost for GMLS or RBF methods is $O(nN)$. The cost for weight generation and evaluation stages are shown by Table 3.1 and Table 3.2.

RBF Algorithm	**Comput. Cost (RBF)**
RBF Interp. Weight Gen.	$O(N_{eval}(n_{rbf} + L)^3)$
RBF-FD Weight Gen.	$O(N(n_{rbf} + L)^3)$
RBF Interp. Eval.	$O(n_{rbf} N_{eval})$
RBF-FD Eval.	$O(n_{rbf} N)$

Table 3.1: Cost of RBF Algorithms [1,2] for interpolation and FD weight generation, and the cost of evaluating interpolant and the approximation of the surface Laplacian.

GMLS Algorithm	**Comput. Cost (GMLS)**
GMLS Interp. Weight Gen.	$O(N_{eval} n_{gmls} L^2)$
GMLS FD Weight Gen.	$O(N n_{gmls} L^2)$
GMLS Interp. Eval.	$O(n_{gmls} N_{eval})$
GMLS FD Eval.	$O(n_{gmls} N)$

Table 3.2: Cost of GMLS algorithms for interpolation and FD weight generation, and the cost of evaluating interpolant and the approximation of the surface Laplacian.

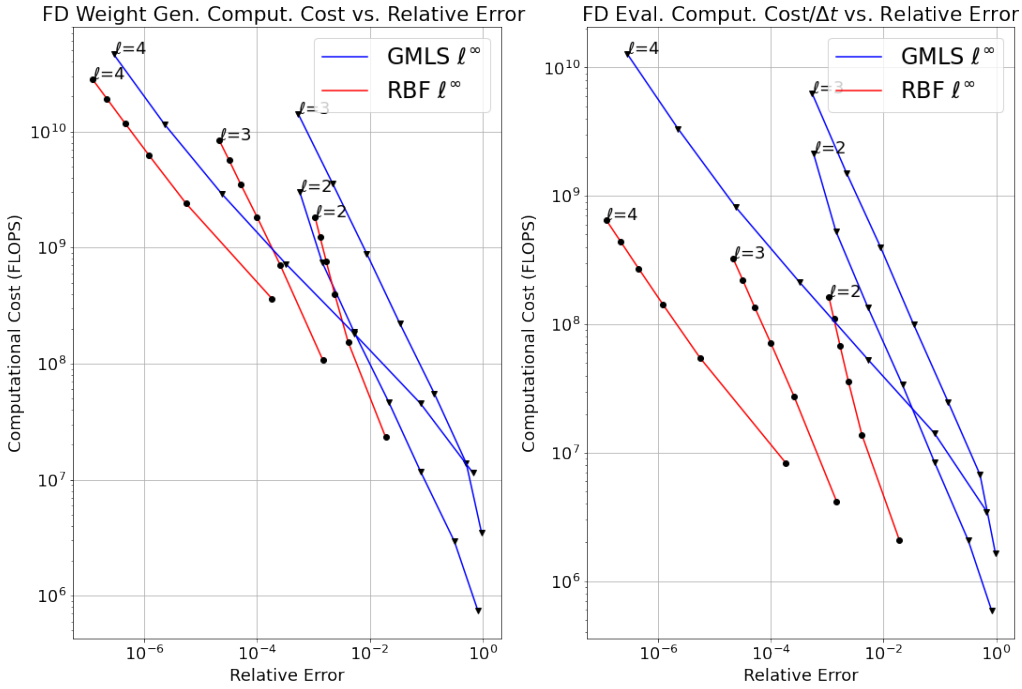


Fig. 3.4: RBF and GMLS algorithms computational cost vs. relative error (Left), and computational cost per timestep for $O(nN)$ vs the relative error (Right). Both plots use convergence results from the cubed sphere node tests.

The evaluation step is cheaper when using RBFs since Compadre's GMLS methods uses larger stencils. The GMLS average stencil sizes are defined by,

$$n_{gmls_{avg}} = \frac{n_{max} + n_{min}}{2},$$

these are shown by Table 3.3.

N	$n_{gmls}, \ell = 2$	$n_{gmls}, \ell = 3$	$n_{gmls}, \ell = 4$
1178	55	92	134
7778	52	93	136
20186	52	90	131
38402	52	90	131
62426	52	90	129
92258	52	87	130

Table 3.3: The average stencil sizes for GMLS with increasing polynomial degree $\ell = 2, 3, 4$ and increasing problem sizes N.

4. Conclusions. For this work we set out to accomplish the following goals:

- Evaluate the interpolant and surface Laplacian of the function $Y_5^4(\boldsymbol{x})$ using RBF methods on icosahedral and cubed sphere nodes.

- Achieve lower relative ℓ^∞ error than GMLS with at least an order of magnitude less nodes. This is shown by Figs. 3.2,3.3.
- Compare the algorithmic costs of RBF methods vs. GMLS. In our initial tests the stencils satisfy $n_{rbf} < n_{gmls}$, which implies the lower cost. However, more investigation is necessary to determine how generally this result may be applied.

The future goals of this work are to include RbfKokkos in the Compadre [8] software package, and apply its methods to various multiphysics applications.

REFERENCES

[1] L. Demanet, *Painless, highly accurate discretizations of the Laplacian on a smooth manifold*, tech. rep., 2006.

[2] G. Fasshauer and J. Zhang, *On choosing "optimal" shape parameters for RBF approximation*, Numerical Algorithms, 45 (2007), pp. 345–368.

[3] N. Flyer and E. Lehto, *Rotational transport on a sphere: Local node refinement with radial basis functions*, Journal of Computational Physics, 229 (2010), pp. 1954 – 1969.

[4] N. Flyer and G. B. Wright, *A radial basis function method for the shallow water equations on a sphere*, Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences, 465 (2009), pp. 1949–1976.

[5] B. Fornberg and N. Flyer, *A Primer on Radial Basis Functions with Applications to the Geosciences*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2015.

[6] Gishi, *Kdtree-gishi*. https://github.com/gishi523/kd-tree.

[7] R. L. Hardy, *Multiquadric equations of topography and other irregular surfaces*, Journal of Geophysical Research (1896-1977), 76 (1971), pp. 1905–1915.

[8] P. Kuberry, P. Bosler, and N. Trask, *Compadre Toolkit*, Zenodo, June 2020.

[9] E. Lehto, V. Shankar, and G. Wright, *A radial basis function (RBF) compact finite difference (FD) scheme for reaction-diffusion*, SIAM Journal on Scientific Computing, 39 (2017), pp. A2129–A2151.

[10] D. Mirzaei, R. Schaback, and M. Dehghan, *On generalized moving least squares and diffuse derivatives*, IMA Journal of Numerical Analysis, 32 (2012), pp. 983–1000.

[11] V. Shankar, G. B. Wright, R. M. Kirby, and A. L. Fogelson, *A radial basis function (RBF)-finite difference (fd) method for diffusion and reaction-diffusion equations on surfaces*, 2014.

[12] S. B. Shaw, *Radial basis function finite difference approximations of the laplace beltrami operator*, Master's thesis, Boise State University, USA, 2019. 1587.

Appendix A. Algorithms. Algorithms 1 and 2 summarize the steps to compute RBF interpolant and RBF-FD weights, respectively, using the Tangent Plane Method (TPM).

Algorithm 1 Method for Computing RBF Interpolant using Tangent Plane Method (TPM).

Select point-list X_D, X_E in dimension d and number of points N
Initialize spline/polynomial degree (ℓ), min-stencil-size (n_{min})
Scale n_{min} by a constant $\alpha \geq 1$
Set stencil-size $n = \alpha n_{min}$
Build KDTree from point-list X_D
for points $\boldsymbol{x}_{e_i}$ in point-list X_E **do**
    Set stencil center $\boldsymbol{x}_{e_c}$ sequentially from point-list X_E
    Find neighbor-list of length stencil-size for $\boldsymbol{x}_{e_c}$ using KDTree of X_D
    Set stencil point-list X_n using neighbor-list
    Set stencil for input function or data f_n using neighbor-list
    Compute QR via Householder reflections on $W = I - \boldsymbol{n}\boldsymbol{n}^T$
    Project and rotate X_n using R
    Construct degree ℓ RBF distance matrix A and bivariate polynomial matrix P; Construct saddle point matrix $\mathcal{A}$ using previous line
    Compute $\phi|_{\boldsymbol{x}=\boldsymbol{x}_{e_c}}$ and $p|_{\boldsymbol{x}=\boldsymbol{x}_{e_c}}$
    Construct right hand side of saddle point system $\boldsymbol{f}$ using previous line
    Solve $\mathcal{A}\boldsymbol{w} = \boldsymbol{f}$ to find weights $\boldsymbol{w}^T$ for stencil centered at $\boldsymbol{x}_c$
    Compute interpolant s_k at x_{e_c} by $\boldsymbol{w} \cdot \boldsymbol{f_n}$
end for

Algorithm 2 Method for Computing RBF-FD Weights using Tangent Plane Method (TPM).

Select point-list X in dimension d and number of points N
Initialize spline/polynomial degree (ℓ), min-stencil-size (n_{min})
Scale n_min by a constant $\alpha \geq 1$
Set stencil-size $n = \alpha n_{min}$
Build KDTree from point-list X
for points $\boldsymbol{x}_i$ in point-list X **do**
    Set stencil center $\boldsymbol{x}_c$ sequentially from point-list X
    Find neighbor-list of length stencil-size for $\boldsymbol{x}_c$ using KDTree
    Set stencil point-list X_n using neighbor-list
    Compute QR via Householder reflections of $W = I - \boldsymbol{n}\boldsymbol{n}^T$
    Project and rotate X_n using R
    Construct degree ℓ RBF distance matrix A and bivariate polynomial matrix P;
    Construct saddle point matrix $\mathcal{A}$ using previous line
    Compute $\mathcal{L}\phi|_{\boldsymbol{x}=\boldsymbol{x}_c}$ and $\mathcal{L}p|_{\boldsymbol{x}=\boldsymbol{x}_c}$
    Construct right hand side of saddle point system $\boldsymbol{f}$ using previous line
    Solve $\mathcal{A}\boldsymbol{w} = \boldsymbol{f}$ to find weights $\boldsymbol{w}^T$ for stencil centered at $\boldsymbol{x}_c$
end for

HYBRID MULTILEVEL MONTE CARLO – POLYNOMIAL CHAOS METHOD FOR GLOBAL SENSITIVITY ANALYSIS

MICHAEL MERRITT*, GIANLUCA GERACI†, MIKE ELDRED‡, AND TERESA PORTONE§

Abstract. Uncertainty Quantification for high-fidelity and high-dimensional computational applications can be prohibitively expensive, heightening the need to efficiently characterize, propagate, and rank the uncertain parameters. Multilevel Monte Carlo (MLMC) sampling approaches provide an efficient strategy for handling the uncertainty propagation of expensive applications. On the other end, Polynomial Chaos expansions (PCE) have been routinely used for Global Sensitivity Analysis (GSA) in cases of moderately large dimensionality and whenever the underlying function is smooth. In this work, we propose a hybrid MLMC-PCE method with a focus on GSA. In particular, the use of a MLMC sampling strategy for the computation of the PCE coefficients is expected to extend the applicability of the PCE-based GSA analysis to expensive high-dimensional problems. In order to efficiently couple these two strategies, the MLMC sample allocation problem is formulated to target PCE-derived metrics for GSA, namely the estimator variance for each term of the Sobol' variance decomposition. The idea presented in this manuscript is demonstrated by means of a simple numerical experiment based on an analytical problem for which the estimated Sobol' indices obtained via the novel MLMC-PCE approach are compared to their single fidelity counterparts and the reference solutions.

1. Introduction. Researchers in computational science continue to expand the state of the art in high-fidelity modeling and simulation, including complex multiphysics and multiscale simulations. For uncertainty quantification (UQ), this often leads to the dual challenge of high computational expense and high random dimensionality, as driven by this increasing model complexity. The most important contributors within this large set of uncertain parameters to the output of said models is often poorly understood. In this context, there are a variety of UQ techniques that allow one to understand how uncertainties propagate through the often many layers of these high-fidelity models. Global sensitivity analysis (GSA) aims to accomplish this goal by quantifying the relative contribution of uncertain parameters to uncertainty in the output of a mathematical model [11]. We focus, in particular, on the estimation of Sobol' indices, a variance-based GSA measure named for the work of Ilya Sobol' [13]. Techniques for computing GSA measures rely on high-dimensional integration methods, such as Monte Carlo sampling. These Monte Carlo methods are simple to implement and their asymptotic convergence properties are independent of the dimensionality of the problem. The drawback of MC methods is that their convergence remains slow, requiring a large number of evaluations of the quantity of interest (QoI) [10, 13].

When faced with a model that is computationally expensive to simulate, it is often advantageous to consider a hierarchy of related models with differing levels of fidelity and associated computational cost. This approach has become popular, specifically through the use of multilevel Monte Carlo (MLMC) [4], which aims to accelerate Monte Carlo convergence to an acceptable error in the statistics by leveraging a hierarchy of models parameterized by a single variable, *e.g.* spatial or temporal resolution. These ML models are often found in the context of using a hierarchy of mesh refinement levels when solving a differential equation [1], although they encompass a broader class of models, such as financial models and risk management, biochemical reaction networks, and engineering problems [4]. The concept of multiple model fidelities has been generalized to more heterogeneous model hierarchies, giving rise to the notion of multifidelity UQ methods such as multifidelity Monte Carlo (MFMC) and generalized control variate methods [8, 5].

*Department of Mathematics, North Carolina State University, mbmerrit@ncsu.edu
†Sandia National Laboratories, ggeraci@sandia.gov
‡Sandia National Laboratories, mseldre@sandia.gov
§Sandia National Laboratories, tporton@sandia.gov

The focus of this study is the application of MLMC methods to better estimate variance-based GSA measures via Polynomial Chaos expansions (PCE). Previous efforts to accelerate the computation of Sobol' indices using a hierarchy of models [9, 7], while successful in reducing the computational cost below that of standard single-fidelity MC, still require a separate MC estimator for each term of the Sobol' variance decomposition, causing their cost to scale with parameter dimension. This would limit their utility in large-scale applications with many parameters. It is also worth noting that these strategies do not provide the opportunity to reuse samples between each MC estimator and therefore their computational cost, measured as number of model evaluations, necessarily increases with the number of dimensions.

The hybrid method proposed in this work is intended to fill the gap between the efficiency of PCE-based GSA decomposition, which naturally allows one to link each Sobol' variance decomposition term directly to the relevant PCE coefficients, with the provable advantages of MLMC for numerical integration in high-dimensional spaces. In this case, although the number of coefficients is still expected to increase with problem dimension, the model evaluations required to estimate each PCE coefficient can be shared and the only difference among the coefficient evaluations is introduced by the polynomial basis used for each coefficient integral, and these bases are known *a priori*.

The remainder of this document is organized as follows: the background for an efficient GSA analysis is presented in Section 2, including necessary details on PCE in Section 2.1 and on sampling approaches in Section 2.2. The derivation of the estimators needed to properly formulate the MLMC sample optimization problem are presented in Section 3. Numerical results for a simple analytical test problem are highlighted in Section 4 where MLMC and MC results are compared to high-resolution solutions for the Sobol' variance decomposition terms. Concluding remarks end the manuscript in Section 5.

2. Global Sensitivity Analysis background. There are a variety of approaches used to determine the sensitivity of a model to uncertain parameters, including local and global sensitivity methods, as well as others [12]. In this manuscript we focus on global sensitivity analysis. Global sensitivity analysis aims to describe the sensitivity of a model over the parameter domain, as opposed to at a single point. The primary variance-based GSA measure, the Sobol' index, quantifies the relative contribution of a subset of uncertain parameters to the output variance of a model. Given a model $Q(\boldsymbol{\xi}) : \Xi \to \mathcal{Q} \subseteq \mathbb{R}$ and a set of uncertain parameters, $\boldsymbol{\xi} = (\xi_1, \ldots, \xi_d) \in \Xi \subset \mathbb{R}^d$, the first step of a Sobol' GSA analysis is the so-called ANalysis Of VAriance (ANOVA) decomposition [13] which decomposes the variance $\mathbb{V}ar\,[Q]$ in a sum of conditional contributions

$$\mathbb{V}ar\,[Q] = \sum_{\substack{u \subseteq \{1,2,\ldots,d\} \\ u \neq 0}} \mathcal{S}_u, \tag{2.1}$$

where each conditional variance $\mathcal{S}_u$ is defined as

$$\mathcal{S}_u = \int Q_u^2(\boldsymbol{\xi}_u) \mathrm{d}\boldsymbol{\xi}_u \tag{2.2}$$

and the term Q_u is obtained as

$$Q_u(\boldsymbol{\xi}_u) = \int Q(\boldsymbol{\xi}_{\sim u}) p(\boldsymbol{\xi}_{\sim u}) \mathrm{d}\boldsymbol{\xi}_{\sim u} - \sum_{\substack{v \subset u \\ v \neq u}} Q_v(\boldsymbol{\xi}_v). \tag{2.3}$$

In the previous equation we used the standard notation $\boldsymbol{\xi}_{\sim u}$ to indicate the random vector which include all variables but the ones belonging to u.

One common sensitivity measure for a parameter, or set of parameters, is the Sobol' index [11]

$$\bar{\mathcal{S}}_u = \frac{\mathcal{S}_u}{\mathbb{V}ar[Q]}. \tag{2.4}$$

Similarly, it is also possible to define for any variable its total contribution to the variance as

$$\bar{\mathcal{T}}_i = \frac{\sum_{u \subseteq \{1,\dots,d\}, \, i \in u} \mathcal{S}_u}{\mathbb{V}ar\,[Q]} = \frac{\mathcal{T}_i}{\mathbb{V}ar\,[Q]}. \tag{2.5}$$

From the definitions of $\mathcal{S}_u$ and $\mathcal{T}_i$ it follows that $\sum_u \bar{\mathcal{S}}_u = 1$ whereas $\sum_i \bar{\mathcal{T}}_i \geq 1$.

In practice, Sobol' indices are often estimated by means of Monte Carlo sampling and there are a variety of methods for doing this [10]. The general idea of using a sampling approach for GSA analysis is to compute conditional expected values contributions for evaluating the terms $\mathcal{S}_u$. Another approach to GSA relies on the Polynomial Chaos method, which provides a convenient way of deriving all the conditional variances directly from the knowledge of the PCE coefficients. Polynomial Chaos and its extension to GSA are described in the next sections.

2.1. Polynomial chaos expansions. Let $Q(\boldsymbol{\xi})$ be a scalar-valued function of a random vector $\boldsymbol{\xi} = (\xi_1, \dots, \xi_d)$. Then the truncated polynomial chaos expansion (PCE) of Q is a spectral expansion of $P+1$ terms given as

$$Q_{PC}(\boldsymbol{\xi}) = \sum_{k=0}^{P} \beta_k \Psi_k(\boldsymbol{\xi}), \tag{2.6}$$

where $\{\Psi_k\}, k = 1, \dots, P$, is a family of orthogonal polynomials and β_k is the corresponding PCE coefficient. The kth multivariate orthogonal polynomial Ψ_k is constructed as a tensor product of 1D orthogonal polynomials,

$$\Psi_k(\xi_1, \dots, \xi_d) = \prod_{i=1}^{d} \psi_{a_i^k}(\xi_i), \quad a^k = (a_1^k, \dots, a_d^k),$$

where a_i^k is a multi-index, denoting the degree of the ith 1D polynomial for the kth multivariate polynomial. If we define a maximal total degree p for the family of multivariate orthogonal polynomials, then the full expansion with d variables and total polynomial order p will have $P+1$ terms given as

$$P + 1 = \frac{(p+d)!}{p!d!}. \tag{2.7}$$

A coefficient must be computed for each of these terms in a PCE. The PCE coefficients are deterministic quantities, and under the formulation of the PCE known as nonintrusive spectral projection (NISP) [6], they are defined as

$$\beta_k = \frac{\mathbb{E}[Q(\boldsymbol{\xi})\Psi_k(\boldsymbol{\xi})]}{\mathbb{E}[\Psi_k^2(\boldsymbol{\xi})]}, \tag{2.8}$$

where the denominator of (2.8) is the squared L^2 norm of the kth polynomial basis. The polynomial basis of the PCE is chosen such that the set of component polynomials are

orthogonal with respect to the distribution of $\boldsymbol{\xi}$. For example a normally-distributed $\boldsymbol{\xi}$ corresponds to the family of orthogonal Hermite polynomials, a uniformly-distributed $\boldsymbol{\xi}$ corresponds to the family of orthogonal Legendre polynomials, and so on (see the Weiner-Askey scheme [14]). The norms of a variety of orthogonal polynomial families are known analytically; see [6]. Thus the main challenge in characterizing a PC expansion is in estimating its coefficients β_k for $k = 0, \ldots, P$. A variety of methods exist for accomplishing this task including regression methods, full tensor quadrature, sparse-grid quadrature, and sampling methods [6]. While many quadrature methods can be shown to converge quickly when evaluating low-dimensional integrals of smooth functions, these methods become prohibitively expensive when dealing with high-dimensional and noisy functions [3, 15].

Similarly, computing the PCE coefficients tends to be expensive when the input dimension of the function Q is large, because the number of terms included in the expansion follows the formula (2.7), and even methods based on regression could require the solution of large linear systems with associated issues related to memory requirements and numerical precision. Under these circumstances, namely large input dimension and lack of regularity of the QoI, Monte Carlo may be the only viable method for computing a large number of high-dimensional integrals, especially because the same set of function evaluations can be used to estimate multiple PCE coefficients, since only the basis term Ψ_k changes in Equation (2.8) for different coefficients.

2.1.1. Application of PCE to global sensitivity analysis. Now that we have introduced the PCE, we turn to the question of performing sensitivity analysis. Extending this to Sobol' indices [2], we can express the variance of Q_{PC} as

$$\mathbb{V}ar[Q_{PC}] = \sum_{k=1}^{P} \beta_k^2 \, \mathbb{E}[\Psi_k^2]$$

and we can compare the ANOVA decomposition, Equation (2.1), to the previous expression obtaining

$$\sum_{k=1}^{P} \beta_k^2 \, \mathbb{E}[\Psi_k^2] \approx \sum_{\substack{u \subseteq \{1,2,\ldots,d\} \\ u \neq 0}} \mathcal{S}_u, \tag{2.9}$$

from which it follows that

$$\mathcal{S}_u \approx \sum_{k \in K_u} \beta_k^2 \, \mathbb{E}[\Psi_k^2], \tag{2.10}$$

where K_u represents the set of coefficients k for which the multivariate polynomial only depends on the subset $\boldsymbol{\xi}_u$.

Given this approach, estimating conditional variances or Sobol' indices from PCE coefficients is simply a matter of summing the proper squared coefficients multiplied by the corresponding basis norms. Estimating the total indices is done in a similar manner. Even for GSA, the main task to be preformed within the PCE is the computation of its coefficients, which corresponds to a high-dimensional quadrature problem. Sampling approaches could potentially provide a scalable method for the quadrature, so the computation of coefficients with both MC and MLMC is described in the following section.

2.2. Monte Carlo and multilevel Monte Carlo sampling. For the QoI $Q(\boldsymbol{\xi})$, let Q_L denote an approximation of Q at the highest resolution level. If one wishes to estimate

the expectation $\mathbb{E}[Q_L(\boldsymbol{\xi})]$ using Monte Carlo sampling, they may use the sample average estimator

$$\hat{Q}_L = \frac{1}{N}\sum_{i=1}^{N} Q_L(\boldsymbol{\xi}^{(i)}), \tag{2.11}$$

where N is the number of samples drawn from the joint probability distribution $p(\boldsymbol{\xi})$. Notice the estimator $\hat{Q}_L$ is itself a random variable with its own mean and variance. Further, the finite resolution associated to Q_L introduces an error with respect to Q, and thus the mean-squared error (MSE) of the estimator $\hat{Q}_L$ accounts for this bias with the decomposition

$$\mathbb{E}[(\hat{Q}_L - \mathbb{E}[Q])^2] = \frac{\mathbb{V}ar[\hat{Q}_L]}{N} + (\mathbb{E}[\hat{Q}_L - Q])^2.$$

Thus to improve the quality of a MC estimate, there are two terms to consider: the variance and the bias. In this work we focus on the variance contribution which dictates that the estimator error decays at the convergence rate of $\mathcal{O}(N^{-1/2})$. This slow rate of convergence is a common shortcoming of standard Monte Carlo methods. Several approaches have been proposed to reduce the variance of $\hat{Q}_L$; see for instance [4, 5, 8].

Among the various variance reduction strategies, multilevel Monte Carlo (MLMC) represents the prototypical example and therefore we will focus on it in this work as a starting point, leaving the extension to arbitrary control variate and multifidelity strategies to future investigations. Let $Q_0, Q_1, \ldots, Q_L$ be a hierarchy of models indexed by ℓ, where an increasing ℓ corresponds to an increasing accuracy. Here Q_L is the highest-fidelity model and the goal is to efficiently estimate $\mathbb{E}[Q_L]$ by making use of the lower-level model evaluations. Typically, this approach is advantageous, due to the fact that the cost of evaluating Q_ℓ, denoted C_ℓ, follows the relation $C_0 \leq C_1 \leq \cdots \leq C_L$. Under these conditions, leveraging cheaper lower-level models can greatly reduce the cost of estimating $\mathbb{E}[Q_L]$. Using the linearity of the expectation operator, we observe

$$\mathbb{E}[Q_L] = \mathbb{E}[Q_0] + \mathbb{E}[Q_1] - \mathbb{E}[Q_0] + \cdots + \mathbb{E}[Q_L] - \mathbb{E}[Q_{L-1}] = \sum_{\ell=0}^{L} \mathbb{E}[Q_\ell - Q_{\ell-1}],$$

where $Q_{-1} = 0$. Thus, we are able to estimate the mean of the difference of adjacent levels, and combine these to form an estimate of the mean at the highest level of accuracy. We define $Y_\ell = Q_\ell - Q_{\ell-1}$ and obtain the expression for the MLMC estimator:

$$\hat{Q}_L^{ML} = \sum_{\ell=0}^{L} \hat{Y}_\ell = \sum_{\ell=0}^{L} \frac{1}{N_\ell} \sum_{i=1}^{N_\ell} Q_\ell^{(i)} - Q_{\ell-1}^{(i)}. \tag{2.12}$$

The goal of this approach is to derive an estimator with a reduced variance, while having the same or lower cost. The idea is that when $\mathbb{V}ar[Y_\ell]$ decreases as $\ell \to L$, fewer samples need to be allocated to the higher levels, which are expensive to evaluate.

The MSE of $\hat{Q}_L^{ML}$ can be decomposed in terms of the variance and bias

$$\mathbb{E}\left[(\hat{Q}_L^{ML} - \mathbb{E}[Q])^2\right] = \mathbb{V}ar[\hat{Q}_L^{ML}] + (\mathbb{E}[Q_L - Q])^2. \tag{2.13}$$

The variance of the ML estimator can be expressed as

$$\mathbb{V}ar[\hat{Q}_L^{ML}] = \sum_{\ell=0}^{L} \frac{\mathbb{V}ar[Y_\ell]}{N_\ell}, \tag{2.14}$$

where independent sampling of each Y_ℓ guarantees that the covariance terms among levels in (2.14) are equal to zero (i.e. $\mathbb{C}ov[Y_\ell, Y_{\ell'}] = 0,\ \ell' \neq \ell$). The optimization problem for sample allocation is defined to minimize the total computational cost across levels, while achieving a balance between the deterministic and stochastic error contributions (bias and variance, respectively) in (2.13). Defined in this manner, the optimization problem has a closed-form solution, which can be found in [4].

2.3. MLMC estimation of PCE coefficients. As an example of the MLMC formulation, we will now describe how to obtain an optimal sampling allocation for the kth PCE coefficient β_k. The ML estimator for β_k is

$$\begin{aligned}\hat{\beta}_k &= \frac{1}{\mathbb{E}[\Psi_k^2]} \sum_{\ell=0}^{L} \widehat{Y_\ell \Psi_k} \\ &= \frac{1}{b_k} \sum_{\ell=0}^{L} \frac{1}{N_\ell} \sum_{i=1}^{N_\ell} (Q_\ell^{(i)} - Q_{\ell-1}^{(i)}) \Psi_k^{(i)} \\ &= \frac{1}{b_k} \sum_{\ell=0}^{L} \frac{1}{N_\ell} \sum_{i=1}^{N_\ell} P_{\ell,k}^{(i)}\end{aligned} \tag{2.15}$$

where $P_{\ell,k} = (Q_\ell - Q_{\ell-1})\Psi_k$ and $b_k = \mathbb{E}[\Psi_k^2]$. Moreover, if C_ℓ denotes the cost of each sample at level ℓ, the total cost is given by

$$C_{tot} = \sum_{\ell=0}^{L} N_\ell C_\ell.$$

To formulate the optimization problem, we consider the variance of $\hat{\beta}_k$, where again we enforce independent sampling on each level:

$$\mathbb{V}ar[\hat{\beta}_k] = \frac{1}{b_k} \sum_{\ell=0}^{L} \frac{\mathbb{V}ar[P_{\ell,k}]}{N_\ell}.$$

The optimal sample allocation can be derived by solving the minimization problem

$$\min_{N_0,\ldots,N_L} \sum_{\ell=0}^{L} N_\ell C_\ell + \lambda^2 \left(\mathbb{V}ar[\hat{\beta}_k] - \varepsilon^2 \right), \tag{2.16}$$

where λ^2 is a Lagrange multiplier, $N_\ell C_\ell$ is the total cost of sampling the QoI at the ℓth level, and ε^2 is the bias error that defines the target variance of the estimator within the error balance constraint. The solution to this problem is related to the canonical MLMC optimal sampling strategy, derived in the work of Giles [4].

For a target estimator variance ε^2, the optimal sampling conditions can be expressed in closed form as

$$N_\ell = \lambda \sqrt{\frac{\mathbb{V}ar[P_{\ell,k}]}{b_k\ C_\ell}},$$

where

$$\lambda = \varepsilon^{-2} \sum_{\ell=0}^{L} \frac{\sqrt{\mathbb{V}ar[P_{\ell,k}]\ C_\ell}}{b_k}.$$

This approach can result in a significant reduction in the estimator variance. From a practical standpoint, it is worth noting that the variances necessary to compute the optimal sampling conditions, $\mathbb{V}ar[P_{\ell,k}]$, are not known a priori. As a result, it is necessary to proceed in an iterative fashion until the relevant statistics converge [4].

The goal of this work is not to solely build a PCE surrogate, but rather to use a PCE surrogate for the purpose of GSA. Therefore in the next section we extend the optimal sample allocation problem to target an ensemble to PCE coefficients, and by extension, the Sobol' indices.

3. Theory/Derivation. To extend the MLMC optimization problem to GSA-related metrics, we first consider the variance of the PCE surrogate variance. It is important to note that each estimated coefficient, $\hat{\beta}_k$, is a random variable and therefore the variance obtained through the PCE expansion is now a random variable as well. The goal of this section is to derive the variance of the variance estimator $\mathbb{V}ar\left[Q_{PC}\right]$, where $\mathbb{V}ar\left[Q_{PC}\right]$ also corresponds to the sum of variances, $\mathcal{S}_u$; see Equation (2.1).

The true variance $\mathbb{V}ar\left[Q\right]$ can be approximated via PCE as

$$\mathbb{V}ar[Q_{PC}] = \sum_{k=1}^{P} \hat{\beta}_k^2 \; b_k, \tag{3.1}$$

and if we now consider that each $\hat{\beta}_k$ is a random variable, we can obtain the expression

$$\mathbb{V}ar\left[\sum_{k=1}^{P} \hat{\beta}_k^2 \; b_k\right] = \sum_{k=1}^{P} \mathbb{V}ar[\hat{\beta}_k^2] \; b_k^2 + \sum_{k=1}^{P} \sum_{\substack{z=1 \\ z\neq k}}^{P} b_k b_z \mathbb{C}ov\left[\hat{\beta}_k^2, \hat{\beta}_z^2\right], \tag{3.2}$$

which corresponds to the variance of the variance estimator based on the PCE expansion. This expression, given the presence of shared samples among coefficients, also incorporates the interaction terms between pairs of coefficients. Similar to the variance decomposition, it is now possible to write

$$\begin{aligned} \mathbb{V}ar[\mathcal{S}_u] &= \mathbb{V}ar\left[\sum_{k\in K_u} \hat{\beta}_k^2 \; b_k\right] \\ &= \sum_{k\in K_u} \mathbb{V}ar[\hat{\beta}_k^2] \; b_k^2 + \sum_{k\in K_u} \sum_{\substack{z\in K_u \\ z\neq k}} b_k b_z \mathbb{C}ov\left[\hat{\beta}_k^2, \hat{\beta}_z^2\right]. \end{aligned} \tag{3.3}$$

If one is able to characterize (3.3) in terms of raw moments of the QoIs and the polynomial bases, then the variance of a particular Sobol' index can be approximated without needing to compute multiple realizations, and a sample allocation method can then be derived to minimize any $\mathbb{V}ar[\mathcal{S}_u]$ for a prescribed cost.

3.1. Derivation of variance and covariance terms. We begin by deriving an expression for $\mathbb{V}ar[(\hat{\beta}_k)^2]$ from (3.3) in terms of moments of Q_ℓ, $\ell = 0, \ldots, L$ and Ψ_k. We know from the Central Limit Theorem that each $\hat{\beta}_k$ will be normally distributed with mean given by β_k, if unbiased, and variance given as

$$\mathbb{V}ar[\hat{\beta}_k] = \frac{1}{b_k^2} \sum_{l=0}^{L} \frac{\mathbb{V}ar\left[P_{\ell,k}\right]}{N_\ell}.$$

In general, for a normally-distributed random variable, $X \sim \mathcal{N}(\mu, \sigma^2)$, we have

$$\begin{aligned}\mathbb{V}ar[X^2] &= \mathbb{E}[X^4] - \mathbb{E}[X^2]^2 \\ &= (\mu^4 + 6\mu^2\sigma^2 + 3\sigma^4) - (\mu^2 + \sigma^2)^2 \\ &= 4\mu^2\sigma^2 + 2\sigma^4.\end{aligned}$$

Using this fact, the variance of our estimator can be expressed as

$$\mathbb{V}ar[(\hat{\beta}_k)^2] = 4\mathbb{E}[\hat{\beta}_k]^2\mathbb{V}ar[\hat{\beta}_k] + 2\mathbb{V}ar[\hat{\beta}_k]^2 \tag{3.4}$$

Deriving an expression for the covariance terms in (3.2) will require a different approach, using the bilinearity of the covariance and matching correlated samples of the QoI. We will start by considering a single-level estimator and then move on to the multilevel case. Thus, for the single-level case, the covariance can be expressed as (see Appendix A for details of the derivation)

$$\begin{aligned}\mathbb{C}ov\left[(\hat{\beta}_k)^2, (\hat{\beta}_z)^2\right] &= \mathbb{C}ov\left[\left(\frac{1}{\mathbb{E}[\Psi_k^2]N}\sum_{i=1}^{N} Q^i\Psi_k^i\right)^2, \left(\frac{1}{\mathbb{E}[\Psi_z^2]N}\sum_{i=1}^{N} Q^i\Psi_z^i\right)^2\right] \\ = \frac{1}{\mathbb{E}[\Psi_k^2]^2\mathbb{E}[\Psi_z^2]^2}\Bigg[&\frac{\mathbb{E}[Q^4\Psi_k^2\Psi_z^2] - \mathbb{E}[Q^2\Psi_k^2]\mathbb{E}[Q^2\Psi_z^2]}{N^3} \\ &+ \frac{(2N-2)\ (\mathbb{E}[Q^3\Psi_k^2\Psi_z]\mathbb{E}[Q\Psi_z] - \mathbb{E}[Q^2\Psi_k^2]\mathbb{E}[Q\Psi_z]^2)}{N^3} \\ &+ \frac{(2N-2)\ \left(\mathbb{E}[Q^3\Psi_z^2\Psi_k]\mathbb{E}[Q\Psi_k] - \mathbb{E}[Q\Psi_k]^2\mathbb{E}[Q^2\Psi_z^2]\right)}{N^3} \\ &+ \frac{(2N-2)\ \left(\mathbb{E}[Q^2\Psi_k\Psi_z]^2\right)}{N^3} \\ &+ \frac{4(N-1)(N-2)\ (\mathbb{E}[Q^2\Psi_k\Psi_z]\mathbb{E}[Q\Psi_k]\mathbb{E}[Q\Psi_z])}{N^3} \\ &- \frac{(4N^2-10N+6)\ (\mathbb{E}[Q\Psi_k]^2\mathbb{E}[Q\Psi_z]^2)}{N^3}\Bigg].\end{aligned} \tag{3.5}$$

The multilevel estimator for the covariance with L levels, which will build upon the expression derived for the single-level estimator in the previous section, is presented in the following. First we define

$$\begin{aligned}\hat{P}_{\ell,k} &= \frac{1}{N_\ell}\sum_{i=1}^{N_\ell} P_{\ell,k}^{(i)} \\ &= \frac{1}{N_\ell}\sum_{i=1}^{N_\ell}\left(Q_\ell^{(i)} - Q_{\ell-1}^{(i)}\right)\Psi_k^{(i)} \\ &= \frac{1}{N_\ell}\sum_{i=1}^{N_\ell} Y_\ell^{(i)}\Psi_k^{(i)}\end{aligned} \tag{3.6}$$

as the single level estimator for the discrepancy function $P_{\ell,k} = Y_\ell\Psi_k$ at level ℓ with respect

to the PCE coefficient k. Using this notation, we write the multilevel covariance term as:

$$
\begin{aligned}
\mathbb{C}ov\left[\left(\hat{\beta}_k\right)^2,\left(\hat{\beta}_z\right)^2\right] = \frac{1}{b_k^2 b_z^2}\sum_{\ell=0}^{L}\Bigg[&\frac{\mathbb{E}\left[P_{\ell,k}^2 P_{\ell,z}^2\right]-\mathbb{E}\left[P_{\ell,k}^2\right]\mathbb{E}\left[P_{\ell,z}^2\right]}{N_\ell^3}\\
&+\frac{(2N_\ell-2)\left(\mathbb{E}\left[P_{\ell,k}^2 P_{\ell,z}\right]\mathbb{E}\left[P_{\ell,z}\right]-\mathbb{E}\left[P_{\ell,k}^2\right]\mathbb{E}\left[P_{\ell,z}\right]^2\right)}{N_\ell^3}\\
&+\frac{(2N_\ell-2)\left(\mathbb{E}\left[P_{\ell,z}^2 P_{\ell,k}\right]\mathbb{E}\left[P_{\ell,k}\right]-\mathbb{E}\left[P_{\ell,z}^2\right]\mathbb{E}\left[P_{\ell,k}\right]^2\right)}{N_\ell^3}\\
&+\frac{(2N_\ell-2)\,\mathbb{E}\left[P_{\ell,k}^2 P_{\ell,z}\right]^2}{N_\ell^3}\\
&+\frac{4(N_\ell-1)(N_\ell-2)\left(\mathbb{E}\left[P_{\ell,k}P_{\ell,z}\right]\mathbb{E}\left[P_{\ell,k}\right]\mathbb{E}\left[P_{\ell,z}\right]\right)}{N_\ell^3}\\
&-\frac{(4N_\ell^2-10N_\ell+6)\left(\mathbb{E}\left[P_{\ell,k}\right]^2\mathbb{E}\left[P_{\ell,z}\right]^2\right)}{N_\ell^3}\\
&+\frac{2}{N_\ell^2}\sum_{\substack{r=0\\r\neq\ell}}^{L}\mathbb{E}\left[P_{\ell,k}^2 P_{\ell,z}\right]\mathbb{E}\left[P_{r,z}\right]-\mathbb{E}\left[P_{\ell,k}^2\right]\mathbb{E}\left[P_{\ell,z}\right]\mathbb{E}\left[P_{r,z}\right]\\
&+2\left(N_\ell-1\right)\left(\mathbb{E}\left[P_{r,z}\right]\mathbb{E}\left[P_{\ell,z}P_{\ell,k}\right]\mathbb{E}\left[P_{\ell,k}\right]-\mathbb{E}\left[P_{r,z}\right]\mathbb{E}\left[P_{\ell,z}\right]\mathbb{E}\left[P_{\ell,k}\right]^2\right)\\
&+\mathbb{E}\left[P_{\ell,z}^2 P_{\ell,k}\right]\mathbb{E}\left[P_{r,k}\right]-\mathbb{E}\left[P_{\ell,z}^2\right]\mathbb{E}\left[P_{\ell,k}\right]\mathbb{E}\left[P_{r,k}\right]\\
&+2\left(N_\ell-1\right)\left(\mathbb{E}\left[P_{r,k}\right]\mathbb{E}\left[P_{\ell,k}P_{\ell,z}\right]\mathbb{E}\left[P_{\ell,z}\right]-\mathbb{E}\left[P_{r,k}\right]\mathbb{E}\left[P_{\ell,k}\right]\mathbb{E}\left[P_{\ell,z}\right]^2\right)\\
&+\frac{1}{N_\ell}\sum_{r=\ell+1}^{L}\frac{4}{N_r}\left(\mathbb{E}\left[P_{\ell,k}P_{\ell,z}\right]\mathbb{E}\left[P_{r,k}P_{r,z}\right]-\mathbb{E}\left[P_{\ell,k}\right]\mathbb{E}\left[P_{\ell,z}\right]\mathbb{E}\left[P_{r,k}\right]\mathbb{E}\left[P_{r,z}\right]\right.\\
&+\left(N_r-1\right)\left(\mathbb{E}\left[P_{\ell,k}P_{\ell,z}\right]\mathbb{E}\left[P_{r,k}\right]\mathbb{E}\left[P_{r,z}\right]-\mathbb{E}\left[P_{\ell,k}\right]\mathbb{E}\left[P_{\ell,z}\right]\mathbb{E}\left[P_{r,k}\right]\mathbb{E}\left[P_{r,z}\right]\right)\\
&+\left.\left(N_\ell-1\right)\left(\mathbb{E}\left[P_{\ell,k}\right]\mathbb{E}\left[P_{\ell,z}\right]\mathbb{E}\left[P_{r,k}P_{r,z}\right]-\mathbb{E}\left[P_{\ell,k}\right]\mathbb{E}\left[P_{r,k}\right]\mathbb{E}\left[P_{r,z}\right]\mathbb{E}\left[P_{\ell,z}\right]\right)\right)\\
&+\sum_{r=\ell+1}^{L}\frac{4}{N_\ell}\sum_{\substack{q=0\\q\neq\ell,r}}^{L}\left(\mathbb{E}\left[P_{\ell,k}P_{\ell,z}\right]\mathbb{E}\left[P_{r,k}\right]\mathbb{E}\left[P_{q,z}\right]-\mathbb{E}\left[P_{\ell,k}\right]\mathbb{E}\left[P_{\ell,z}\right]\mathbb{E}\left[P_{r,k}\right]\mathbb{E}\left[P_{q,z}\right]\right)\\
&+\frac{4}{N_r}\sum_{\substack{q=0\\q\neq\ell,r}}^{L}\left(\mathbb{E}\left[P_{\ell,k}P_{r,z}\right]\mathbb{E}\left[P_{r,k}\right]\mathbb{E}\left[P_{q,z}\right]-\mathbb{E}\left[P_{\ell,k}\right]\mathbb{E}\left[P_{r,z}\right]\mathbb{E}\left[P_{r,k}\right]\mathbb{E}\left[P_{q,z}\right]\right)\Bigg].
\end{aligned}
\tag{3.7}
$$

The derivation for the above term is reported in detail in Appendix B. We have expressions for the multilevel variance and covariance, expressed in terms of moments of our multilevel QoI and the orthogonal polynomials. We can then use Equation (3.3) to arrive at an estimate of the variance of the conditional variances, $\mathcal{S}_u$.

Next, we use this composition for Equation (3.3) to derive an optimal sample allocation by either minimizing estimator variance subject to an aggregate cost constraint, or alternatively minimizing cost subject to a prescribed level of estimator variance. Here, we adopt the former approach to determine the sample profile $(N_0, N_1, \ldots, N_L)$ that minimizes the stochastic error within a prescribed computational budget:

$$
\min_{N_0,\ldots,N_L}\mathbb{V}ar[\mathcal{S}_u] \quad \text{subject to} \quad \sum_{\ell=0}^{L}N_\ell C_\ell \leq \bar{C}, \quad 0 \leq N_0,\ldots,N_L, \tag{3.8}
$$

where $\bar{C}$ is some upper limit on the total cost of the ML estimator. The choice to minimize

the variance subject to a cost constraint versus the classical approach of minimizing the cost subject to a variance constraint [4] comes down to a matter of practicality. It is more likely that a user of this method will have a limited computational budget in mind as a target, rather than a target accuracy for a particular conditional variance term, which may be difficult to determine prior to performing GSA.

For this study, we will solve (3.8) numerically, using SciPy's optimization with the Sequential Least Squares Programming (SLSQP) algorithm. In the next section, we present preliminary results illustrating the results of this method.

4. Numerical results. The following results are obtained using the Ishigami function, a standard test problem in the GSA literature [2], adapted here to include three levels of approximation with an increasing cost.

4.1. Single-level results. The single-level Ishigami function is

$$Q = \sin(\xi_1) + a\sin^2(\xi_2) + b\xi_3^4\sin(\xi_1), \tag{4.1}$$

where ξ_1, ξ_2, and ξ_3 are uncertain parameters following a uniform distribution on $[-\pi, \pi]$, with a and b being constants. Using multivariate Legendre polynomials, we will compute a single-level PCE of (4.1) where we let $a = 7$ and $b = 0.1$.

We first consider the variability of the estimators $\hat{\beta}_k$, for the first 50 PCE coefficients computed using 100 samples. We compute 2000 replicates of the set of PCE coefficients and then plot the mean and two standard deviations as error bars for each coefficient below.

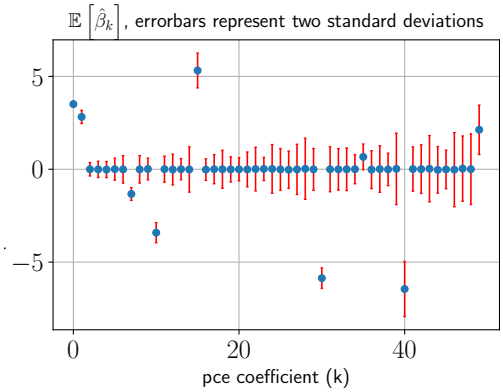


Fig. 4.1: Mean Ishigami PCE coefficients $k = 0, \ldots, 49$, error bars denote 2 standard deviations.

Figure 4.1 shows the increasing variance of the PCE coefficients with the polynomial order, which is to be expected. As the order of an orthogonal polynomial increases, it will also result in a larger variance.

4.2. Multilevel results. Continuing to the multilevel form of the Ishigami function, we define a three level model hierarchy, with associated costs, as follows:

$$\begin{aligned} Q &= \sin(\xi_1) + a\sin^2(\xi_2) + bx_3^4\sin(\xi_1) \\ Q_0 &: a = (0.6)7.0,\ b = (0.6)0.1,\ C_0 = 1 \\ Q_1 &: a = (0.8)7.0,\ b = (0.8)0.1,\ C_1 = 10 \\ Q_2 &: a = (1.0)7.0,\ b = (1.0)0.1,\ C_2 = 100 \end{aligned} \tag{4.2}$$

We can then compute a ML estimate of the PCE coefficients, using the method described in Section 2.3, from the estimator

$$\hat{\beta}_k = \frac{1}{b_k} \sum_{\ell=0}^{L} \frac{1}{N_\ell} \sum_{i=1}^{N_\ell} P_{\ell,k}^{(i)} = \frac{1}{b_k} \sum_{\ell=0}^{L} \frac{1}{N_\ell} \sum_{i=1}^{N_\ell} (Q_\ell^{(i)} - Q_{\ell-1}^{(i)}) \Psi_k^{(i)}. \tag{4.3}$$

As we have shown in Section 3.1, we are able to propagate the uncertainty in the estimated PCE coefficients (4.3) through to the conditional variances, resulting in a new derivation for $\mathbb{V}ar[\mathcal{S}_u]$. To verify the results of the derivation in Section 3.1, we compute 1000 independent realizations of the conditional variances $(\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3, \mathcal{S}_{1,2}, \mathcal{S}_{1,3}, \mathcal{S}_{2,3}, \mathcal{S}_{1,2,3})$ for the Ishigami function. To do this, we compute 1000 realizations of the full set of PCE coefficients up to a total polynomial order of 5. For each realization, we estimate the conditional terms $\mathcal{S}_u$ and $\mathcal{T}_i$ along with their predicted variance. The result of this computation is one estimate of all $\mathbb{V}ar[\mathcal{S}_u]$ and $\mathbb{V}ar[\mathcal{T}_i]$, computed from the 1000 realizations, along with 1000 estimations for $\mathbb{V}ar[\mathcal{S}_u]$ and $\mathbb{V}ar[\mathcal{T}_i]$, computed using the relationships derived in the previous section. The results of this numerical experiment are reported in Fig. 4.2.

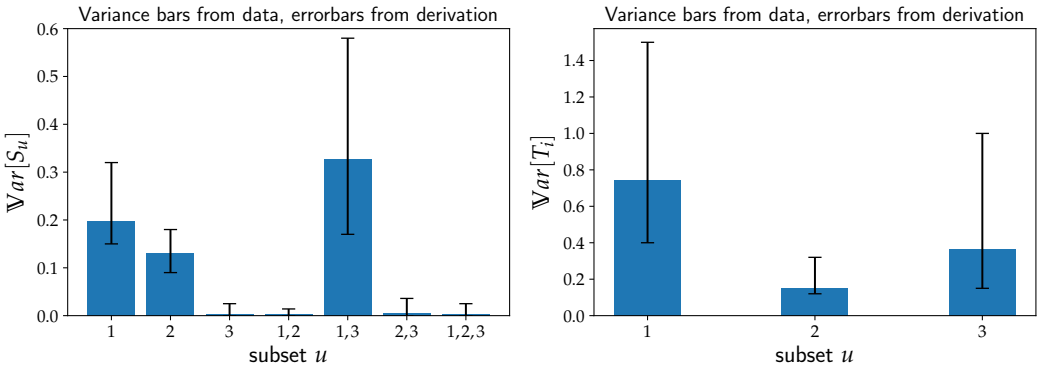


Fig. 4.2: Blue bars indicate the variance of conditional variances from the data. Black error bars represent a 2 standard deviation interval of the estimated variance from the derivation.

Given a valid estimator for the variance of a set of Sobol' indices and a ML estimator for the PCE coefficients, we turn to the issue of verifying the optimal sample profile described by (3.8). To demonstrate the advantages of this approach, we compare the accuracy of the conditional variances computed from evaluating the single-level function only and by using the MLMC estimator with optimal sampling. Figure 5.1 shows the PDFs of each conditional variance computed using 1000 realizations of both the SL and optimal MLMC methods, where the optimal MLMC targets the conditional variance $\mathcal{S}_1$. We note that our approach offers flexibility in determining which conditional term or set of terms we want to target. To properly resolve each conditional contribution, we estimate all PCE coefficients up to a total polynomial order of 8. The set of PDFs below corresponds to SL and ML estimators with the same evaluation cost, corresponding to 500 evaluations of the single-level estimator. Finally, we include a reference value for each conditional variance, which corresponds to the PCE truncation level we use.

The ML sample profile determined by the optimization routine is given by: $[N_0, N_1, N_2] = [12447, 784, 264]$. We noticed a large variability in the solution returned by the optimizer, dependent on the initial guess and resolution level of the statistical moments needed to evaluate the objective function of (3.8). This leads us to speculate about the existence of

multiple local minima and possible noise being present in the objective function entering the optimization. With this in mind, it is likely that the sample profile obtained for this problem is suboptimal. This point requires further study in future work. We do, however, stress the point that even this suboptimal sample profile results in a ML estimator for a given Sobol' index that is more accurate and has a lower variance than the corresponding SL estimator in every case we observe. Further study will require an investigation of the optimization problem and the resulting solutions. An ideal situation would be to derive the optimal sample allocation in closed form, which is a feature of MLMC methods focused on the mean estimator.

5. Conclusion and perspectives. In this work, we explored a hybrid MLMC-PCE approach for GSA that leverages the ANOVA decomposition traditionally used with PCE, but for which the polynomial coefficients are evaluated by means of MLMC. This hybrid approach enables reduction of overall cost by fusing information from multiple sources with distinct accuracy and cost. We focused on developing the main components of the algorithm and presenting preliminary numerical results that demonstrate how the hybrid approach is mathematically sound. In the continuation of the work, we plan to pursue a comprehensive numerical campaign to test the trade-off of this hybrid algorithm when compared with traditional PCE counterparts as a function of input dimension, function regularity, and the presence of noisy data.

REFERENCES

[1] K. A. Cliffe, M. B. Giles, R. Scheichl, and A. L. Teckentrup, *Multilevel Monte Carlo methods and applications to elliptic PDEs with random coefficients*, Computing and Visualization in Science, 14 (2011), p. 3.

[2] T. Crestaux, O. Le Maitre, and J.-M. Martinez, *Polynomial chaos expansion for sensitivity analysis*, Reliability Engineering & System Safety, 94 (2009), pp. 1161–1172.

[3] J. Dick, F. Y. Kuo, and I. H. Sloan, *High-dimensional integration: the quasi-monte carlo way*, Acta Numerica, 22 (2013), p. 133.

[4] M. B. Giles, *Multilevel Monte Carlo methods*, Acta Numerica, 24 (2015), p. 259.

[5] A. A. Gorodetsky, G. Geraci, M. S. Eldred, and J. D. Jakeman, *A generalized approximate control variate framework for multifidelity uncertainty quantification*, Journal of Computational Physics, 408 (2020), p. 109257.

[6] O. Le Maître and O. M. Knio, *Spectral methods for uncertainty quantification: with applications to computational fluid dynamics*, Springer Science & Business Media, 2010.

[7] P. Mycek and M. De Lozzo, *Multilevel Monte Carlo covariance estimation for the computation of Sobol' indices*, SIAM/ASA Journal on Uncertainty Quantification, 7 (2019), pp. 1323–1348.

[8] B. Peherstorfer, K. Willcox, and M. Gunzburger, *Survey of multifidelity methods in uncertainty propagation, inference, and optimization*, Siam Review, 60 (2018), pp. 550–591.

[9] E. Qian, B. Peherstorfer, D. O'Malley, V. V. Vesselinov, and K. Willcox, *Multifidelity monte carlo estimation of variance and sensitivity indices*, SIAM/ASA Journal on Uncertainty Quantification, 6 (2018), pp. 683–706.

[10] A. Saltelli, P. Annoni, I. Azzini, F. Campolongo, M. Ratto, and S. Tarantola, *Variance based sensitivity analysis of model output. Design and estimator for the total sensitivity index*, Computer physics communications, 181 (2010), pp. 259–270.

[11] A. Saltelli, M. Ratto, T. Andres, F. Campolongo, J. Cariboni, D. Gatelli, M. Saisana, and S. Tarantola, *Global sensitivity analysis: the primer*, John Wiley & Sons, 2008.

[12] R. C. Smith, *Uncertainty quantification: theory, implementation, and applications*, vol. 12, Siam, 2013.

[13] I. M. Sobol, *Global sensitivity indices for nonlinear mathematical models and their Monte Carlo estimates*, Mathematics and computers in simulation, 55 (2001), pp. 271–280.

[14] D. Xiu and G. E. Karniadakis, *The Wiener–Askey polynomial chaos for stochastic differential equations*, SIAM journal on scientific computing, 24 (2002), pp. 619–644.

[15] Z. Zhang, M. V. Tretyakov, B. Rozovskii, and G. E. Karniadakis, *A recursive sparse grid collocation method for differential equations with white noise*, SIAM Journal on Scientific Computing, 36 (2014), pp. A1652–A1677.

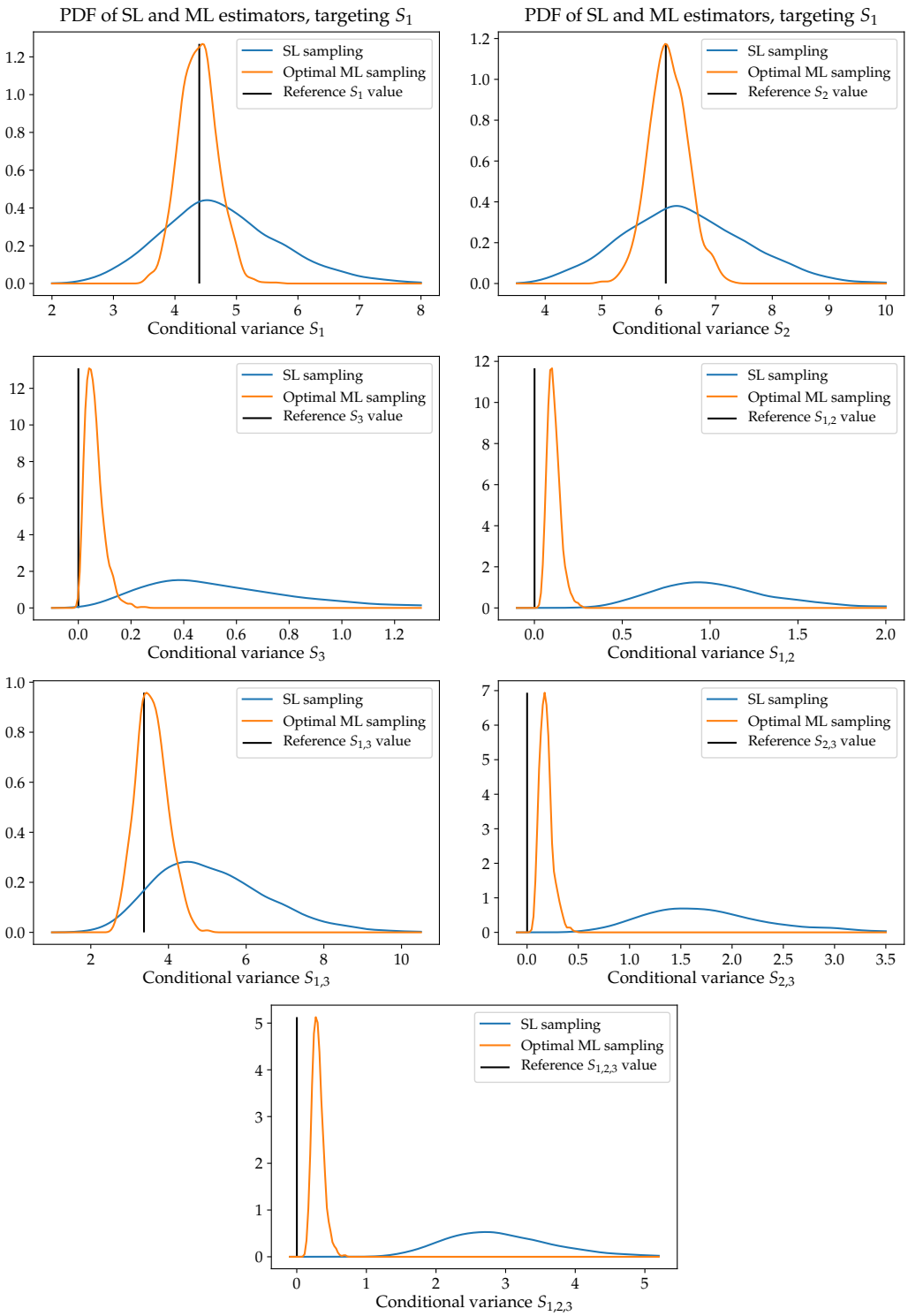


Fig. 5.1: PDFs of SL and optimal ML estimators for each main effect index. Optimal ML allocation targets $\mathbb{V}ar[\mathcal{S}_1]$ in each figure. Black lines indicate reference for each conditional variance.

Appendix A. Single level covariance $\mathbb{C}ov\left[\left(\hat{\beta}_k\right)^2, \left(\hat{\beta}_z\right)^2\right]$**.**

Proof. Letting $P_k^{(i)} = Q^{(i)}\Psi_k^{(i)}$ and $b_k = \mathbb{E}\left[\Psi_k^2\right]$, we have

$$\begin{aligned}
\mathbb{C}ov\left[\left(\hat{\beta}_k\right)^2, \left(\hat{\beta}_z\right)^2\right] &= \frac{1}{N^4 b_k^2 b_z^2}\mathbb{C}ov\left[\left(\sum_{i=1}^N P_k^{(i)}\right)^2, \left(\sum_{i=1}^N P_z^{(i)}\right)^2\right] \\
&= \frac{1}{N^4 b_k^2 b_z^2}\mathbb{C}ov\left[\sum_{i=1}^N P_k^{2,(i)} + \sum_{i=1}^N P_k^{(i)}\sum_{\substack{j=1\\j\neq i}}^N P_k^{(j)}, \sum_{i=1}^N P_z^{2,(i)} + \sum_{i=1}^N P_z^{(i)}\sum_{\substack{j=1\\j\neq i}}^N P_z^{(j)}\right] \\
&= \frac{1}{N^4 b_k^2 b_z^2}\left(\mathbb{C}ov\left[\sum_{i=1}^N P_k^{2,(i)}, \sum_{i=1}^N P_z^{2,(i)}\right] + \mathbb{C}ov\left[\sum_{i=1}^N P_k^{2,(i)}, \sum_{i=1}^N P_z^{(i)}\sum_{\substack{j=1\\j\neq i}}^N P_z^{(j)}\right]\right. \\
&\left. + \mathbb{C}ov\left[\sum_{i=1}^N P_k^{(i)}\sum_{\substack{j=1\\j\neq i}}^N P_k^{(j)}, \sum_{i=1}^N P_z^{2,(i)}\right] + \mathbb{C}ov\left[\sum_{i=1}^N P_k^{(i)}\sum_{\substack{j=1\\j\neq i}}^N P_k^{(j)}, \sum_{i=1}^N P_z^{(i)}\sum_{\substack{j=1\\j\neq i}}^N P_z^{(j)}\right]\right).
\end{aligned} \tag{A.1}$$

We will now consider all the covariance contributions separately. The first term is $\mathbb{C}ov\left[\sum_{i=1}^N P_k^{2,(i)}, \sum_{i=1}^N P_z^{2,(i)}\right]$:

$$\mathbb{C}ov\left[\sum_{i=1}^N P_k^{2,(i)}, \sum_{i=1}^N P_z^{2,(i)}\right] = N\mathbb{C}ov\left[P_k^2, P_z^2\right] = N\left(\mathbb{E}\left[Q^4\Psi_k^2\Psi_z^2\right] - \mathbb{E}\left[Q^2\Psi_k^2\right]\mathbb{E}\left[Q^2\Psi_z^2\right]\right) \tag{A.2}$$

The second contribution we consider is $\mathbb{C}ov\left[\sum_{i=1}^N P_k^{2,(i)}, \sum_{i=1}^N P_z^{(i)}\sum_{\substack{j=1\\j\neq i}}^N P_z^{(j)}\right]$:

$$\begin{aligned}
\mathbb{C}ov\left[\sum_{i=1}^N P_k^{2,(i)}, \sum_{i=1}^N P_z^{(i)}\sum_{\substack{j=1\\j\neq i}}^N P_z^{(j)}\right] &= \mathbb{C}ov\left[\sum_{i=1}^N P_k^{2,(i)}, P_z^{(i)}\sum_{\substack{j=1\\j\neq i}}^N P_z^{(j)} + \sum_{\substack{q=1\\q\neq i}}^N P_z^{(q)}\sum_{\substack{j=1\\j\neq q}}^N P_z^{(j)}\right] \\
&= \mathbb{C}ov\left[\sum_{i=1}^N P_k^{2,(i)}, P_z^{(i)}\sum_{\substack{j=1\\j\neq i}}^N P_z^{(j)} + P_z^{(i)}\sum_{\substack{q=1\\q\neq i}}^N P_z^{(q)} + \sum_{\substack{q=1\\q\neq i}}^N P_z^{(q)}\sum_{\substack{j=1\\j\neq q,i}}^N P_z^{(j)}\right] \\
&= \mathbb{C}ov\left[\sum_{i=1}^N P_k^{2,(i)}, 2P_z^{(i)}\sum_{\substack{j=1\\j\neq i}}^N P_z^{(j)} + \sum_{\substack{q=1\\q\neq i}}^N P_z^{(q)}\sum_{\substack{j=1\\j\neq q,i}}^N P_z^{(j)}\right] \\
&= 2N(N-1)\mathbb{C}ov\left[P_k^2, P_z P_z'\right] \\
&= 2N(N-1)\left(\mathbb{E}\left[Q^3\Psi_k^2\Psi_z\right]\mathbb{E}\left[Q\Psi_z\right] - \mathbb{E}\left[Q^2\Psi_k^2\right]\mathbb{E}\left[Q\Psi_z\right]^2\right),
\end{aligned} \tag{A.3}$$

where P_z and P_z' indicate i.i.d. realizations of P_z.

For symmetry, the third term is simply

$$\mathbb{C}ov\left[\sum_{i=1}^{N} P_z^{2,(i)}, \sum_{i=1}^{N} P_k^{(i)} \sum_{\substack{j=1 \\ j\neq i}}^{N} P_k^{(j)}\right] = 2N(N-1)\left(\mathbb{E}\left[Q^3\Psi_z^2\Psi_k\right]\mathbb{E}\left[Q\Psi_k\right] - \mathbb{E}\left[Q^2\Psi_z^2\right]\mathbb{E}\left[Q\Psi_k\right]^2\right). \tag{A.4}$$

The last contribution is obtained as

$$\begin{aligned}
&\mathbb{C}ov\left[\sum_{i=1}^{N} P_k^{(i)} \sum_{\substack{j=1 \\ j\neq i}}^{N} P_k^{(j)}, \sum_{i=1}^{N} P_z^{(i)} \sum_{\substack{j=1 \\ j\neq i}}^{N} P_z^{(j)}\right] \\
&= 2\mathbb{C}ov\left[\sum_{i=1}^{N} P_k^{(i)} \sum_{j>1}^{N} P_k^{(j)}, \sum_{i=1}^{N} P_z^{(i)} \sum_{\substack{j=1 \\ j\neq i}}^{N} P_z^{(j)}\right] \\
&= 2\mathbb{C}ov\left[\sum_{i=1}^{N}\sum_{j>1}^{N} P_k^{(i)}P_k^{(j)}, P_z^{(i)} \sum_{\substack{j=1 \\ j\neq i}}^{N} P_z^{(j)} + \sum_{\substack{q=1 \\ q\neq i}}^{N} P_z^{(q)} \sum_{\substack{j=1 \\ j\neq q}}^{N} P_z^{(j)}\right] \\
&= 2\mathbb{C}ov\left[\sum_{i=1}^{N}\sum_{j>1}^{N} P_k^{(i)}P_k^{(j)}, P_z^{(i)} \left(P_z^{(j)} + \sum_{\substack{q=1 \\ q\neq i,j}}^{N} P_z^{(q)}\right) + \sum_{\substack{q=1 \\ q\neq i}}^{N} P_z^{(q)} \sum_{\substack{j=1 \\ j\neq q}}^{N} P_z^{(j)}\right] \\
&= 2\mathbb{C}ov\Bigg[\sum_{i=1}^{N}\sum_{j>1}^{N} P_k^{(i)}P_k^{(j)}, P_z^{(i)} \left(P_z^{(j)} + \sum_{\substack{q=1 \\ q\neq i,j}}^{N} P_z^{(q)}\right) \\
&+ P_z^{(j)} \left(P_z^{(i)} + \sum_{\substack{r=1 \\ r\neq j,i}}^{N} P_z^{(r)}\right) + \sum_{\substack{q=1 \\ q\neq i,j}}^{N} P_z^{(q)} \left(P_z^{(i)} + P_z^{(j)} + \sum_{\substack{r=1 \\ r\neq q,i,j}}^{N} P_z^{(r)}\right)\Bigg] \\
&= 2\mathbb{C}ov\left[\sum_{i=1}^{N}\sum_{j>1}^{N} P_k^{(i)}P_k^{(j)}, 2P_z^{(i)}P_z^{(j)} + 2P_z^{(i)} \sum_{\substack{q=1 \\ q\neq i,j}}^{N} P_z^{(q)} + 2P_z^{(j)} \sum_{\substack{r=1 \\ r\neq j,i}}^{N} P_z^{(r)} + \sum_{\substack{q=1 \\ q\neq i,j}}^{N} P_z^{(q)} \sum_{\substack{r=1 \\ r\neq q,i,j}}^{N} P_z^{(r)}\right] \\
&= 4\frac{N(N-1)}{2}\mathbb{C}ov\left[P_kP_k', P_zP_z'\right] + 4\frac{N(N-1)(N-2)}{2}\mathbb{C}ov\left[P_kP_k', P_zP_z''\right] \\
&= 2N(N-1)\left(\mathbb{E}\left[Q^2\Psi_k\Psi_z\right]^2 - \mathbb{E}\left[Q\Psi_k\right]^2\mathbb{E}\left[Q\Psi_z\right]^2\right) \\
&+ 4N(N-1)(N-2)\left(\mathbb{E}\left[Q^2\Psi_k\Psi_z\right]\mathbb{E}\left[Q\Psi_k\right]\mathbb{E}\left[Q\Psi_z\right] - \mathbb{E}\left[Q\Psi_k\right]^2\mathbb{E}\left[Q\Psi_z\right]^2\right) \\
&= 2N(N-1)\mathbb{E}\left[Q^2\Psi_k\Psi_z\right]^2 + 4N(N-1)(N-2)\mathbb{E}\left[Q^2\Psi_k\Psi_z\right]\mathbb{E}\left[Q\Psi_k\right]\mathbb{E}\left[Q\Psi_z\right] \\
&- 2N(N-1)(1+2(N-2))\mathbb{E}\left[Q\Psi_k\right]^2\mathbb{E}\left[Q\Psi_z\right]^2
\end{aligned} \tag{A.5}$$

□

Appendix B. Multilevel MC covariance $\mathbb{C}ov\left[\left(\hat{\beta}_k\right)^2,\left(\hat{\beta}_z\right)^2\right]$. In this section, the derivation of the multilevel covariance term in Equation 3.7 is presented.

Proof. The multilevel MC covariance term can be written as

$$\begin{aligned}
\mathbb{C}ov\left[\left(\hat{\beta}_k\right)^2,\left(\hat{\beta}_z\right)^2\right] &= \mathbb{C}ov\left[\left(\frac{1}{b_k}\sum_{\ell=0}^{L}\hat{P}_{\ell,k}\right)^2,\left(\frac{1}{b_z}\sum_{\ell=0}^{L}\hat{P}_{\ell,z}\right)^2\right]\\
&= \frac{1}{b_k^2 b_z^2}\mathbb{C}ov\left[\left(\sum_{\ell=0}^{L}\hat{P}_{\ell,k}\right)^2,\left(\sum_{\ell=0}^{L}\hat{P}_{\ell,z}\right)^2\right]\\
&= \frac{1}{b_k^2 b_z^2}\mathbb{C}ov\left[\sum_{\ell=0}^{L}\left(\hat{P}_{\ell,k}\right)^2+\sum_{\ell=0}^{L}\sum_{r\neq\ell}\hat{P}_{\ell,k}\hat{P}_{r,k},\sum_{\ell=0}^{L}\left(\hat{P}_{\ell,z}\right)^2+\sum_{\ell=0}^{L}\sum_{r\neq\ell}\hat{P}_{\ell,z}\hat{P}_{r,z}\right]\\
&= \frac{1}{b_k^2 b_z^2}\left(\mathbb{C}ov\left[\sum_{\ell=0}^{L}\left(\hat{P}_{\ell,k}\right)^2,\sum_{\ell=0}^{L}\left(\hat{P}_{\ell,z}\right)^2\right]+\mathbb{C}ov\left[\sum_{\ell=0}^{L}\left(\hat{P}_{\ell,k}\right)^2,\sum_{\ell=0}^{L}\sum_{r\neq\ell}\hat{P}_{\ell,z}\hat{P}_{r,z}\right]\right.\\
&\quad\left.+\,\mathbb{C}ov\left[\sum_{\ell=0}^{L}\left(\hat{P}_{\ell,z}\right)^2,\sum_{\ell=0}^{L}\sum_{r\neq\ell}\hat{P}_{\ell,k}\hat{P}_{r,k}\right]+\mathbb{C}ov\left[\sum_{\ell=0}^{L}\sum_{r\neq\ell}\hat{P}_{\ell,k}\hat{P}_{r,k},\sum_{\ell=0}^{L}\sum_{r\neq\ell}\hat{P}_{\ell,z}\hat{P}_{r,z}\right]\right),
\end{aligned} \tag{B.1}$$

where each single level estimator $\hat{P}_{\ell,k}$ is defined as

$$\hat{P}_{\ell,k}=\frac{1}{N_\ell}\sum_{i=1}^{N_\ell}\left(Q_\ell^{(i)}-Q_{\ell-1}^{(i)}\right)\Psi_k^{(i)}=\frac{1}{N_\ell}\sum_{i=1}^{N_\ell}Y_\ell^{(i)}\Psi_k^{(i)}=\frac{1}{N_\ell}\sum_{i=1}^{N_\ell}P_{\ell,k}^{(i)}. \tag{B.2}$$

There are 4 terms that need to be computed, however for symmetry only 3 of them need to be derived explicitly. The first term can be written as

$$\begin{aligned}
\mathbb{C}ov\left[\sum_{\ell=0}^{L}\left(\hat{P}_{\ell,k}\right)^2,\sum_{\ell=0}^{L}\left(\hat{P}_{\ell,z}\right)^2\right] &= \sum_{\ell=0}^{L}\mathbb{C}ov\left[\left(\hat{P}_{\ell,k}\right)^2,\left(\hat{P}_{\ell,z}\right)^2\right]\\
&= \sum_{\ell=0}^{L}\left[\frac{\mathbb{E}\left[P_{\ell,k}^2P_{\ell,z}^2\right]-\mathbb{E}\left[P_{\ell,k}^2\right]\mathbb{E}\left[P_{\ell,z}^2\right]}{N_\ell^3}\right.\\
&\quad+\frac{(2N_\ell-2)\left(\mathbb{E}\left[P_{\ell,k}^2P_{\ell,z}\right]\mathbb{E}\left[P_{\ell,z}\right]-\mathbb{E}\left[P_{\ell,k}^2\right]\mathbb{E}\left[P_{\ell,z}\right]^2\right)}{N_\ell^3}\\
&\quad+\frac{(2N_\ell-2)\left(\mathbb{E}\left[P_{\ell,z}^2P_{\ell,k}\right]\mathbb{E}\left[P_{\ell,k}\right]-\mathbb{E}\left[P_{\ell,z}^2\right]\mathbb{E}\left[P_{\ell,k}\right]^2\right)}{N_\ell^3}\\
&\quad+\frac{(2N_\ell-2)\,\mathbb{E}\left[P_{\ell,k}^2P_{\ell,z}\right]^2}{N_\ell^3}\\
&\quad+\frac{4(N_\ell-1)(N_\ell-2)\left(\mathbb{E}\left[P_{\ell,k}P_{\ell,z}\right]\mathbb{E}\left[P_{\ell,k}\right]\mathbb{E}\left[P_{\ell,z}\right]\right)}{N_\ell^3}\\
&\quad\left.-\frac{(4N_\ell^2-10N_\ell+6)\left(\mathbb{E}\left[P_{\ell,k}\right]^2\mathbb{E}\left[P_{\ell,z}\right]^2\right)}{N_\ell^3}\right].
\end{aligned} \tag{B.3}$$

For this term the derivation of each single level estimator is identical to the single level covariance term derived in the previous section.

The second (and third) term can be written as

$$\begin{aligned}\mathbb{C}ov\left[\sum_{\ell=0}^{L}\left(\hat{P}_{\ell,k}\right)^2,\sum_{\ell=0}^{L}\sum_{\substack{r=0\\r\neq\ell}}^{L}\hat{P}_{\ell,z}\hat{P}_{r,z}\right]&=\mathbb{C}ov\left[\sum_{\ell=0}^{L}\left(\hat{P}_{\ell,k}\right)^2,2\hat{P}_{\ell,z}\sum_{\substack{r=0\\r\neq\ell}}^{L}\hat{P}_{r,z}+\sum_{\substack{q=0\\q\neq\ell}}^{L}\hat{P}_{q,z}\sum_{\substack{r=0\\r\neq q,\ell}}^{L}\hat{P}_{r,z}\right]\\&=2\sum_{\ell=0}^{L}\mathbb{C}ov\left[\left(\hat{P}_{\ell,k}\right)^2,\hat{P}_{\ell,z}\sum_{\substack{r=0\\r\neq\ell}}^{L}\hat{P}_{r,z}\right]\end{aligned}\tag{B.4}$$

The term $\mathbb{C}ov\left[\left(\hat{P}_{\ell,k}\right)^2,\hat{P}_{\ell,z}\sum_{\substack{r=0\\r\neq\ell}}^{L}\hat{P}_{r,z}\right]$ can be evaluated in term of moments of the QoI Q_ℓ by further manipulating the terms.

$$\begin{aligned}&\mathbb{C}ov\left[\left(\hat{P}_{\ell,k}\right)^2,\hat{P}_{\ell,z}\sum_{\substack{r=0\\r\neq\ell}}^{L}\hat{P}_{r,z}\right]\\&=\sum_{\substack{r=0\\r\neq\ell}}^{L}\mathbb{C}ov\left[\left(\hat{P}_{\ell,k}\right)^2,\hat{P}_{\ell,z}\hat{P}_{r,z}\right]\\&=\sum_{\substack{r=0\\r\neq\ell}}^{L}\mathbb{C}ov\left[\left(\frac{1}{N_\ell}\sum_{i=1}^{N_\ell}P_{\ell,k}^{(i)}\right)^2,\left(\frac{1}{N_\ell}\sum_{i=1}^{N_\ell}P_{\ell,z}^{(i)}\right)\left(\frac{1}{N_r}\sum_{j=1}^{N_r}P_{r,z}^{(j)}\right)\right]\\&=\frac{1}{N_\ell^3}\sum_{\substack{r=0\\r\neq\ell}}^{L}\frac{1}{N_r}\mathbb{C}ov\left[\sum_{i=1}^{N_\ell}\left(P_{\ell,k}^{(i)}\right)^2+\sum_{i=1}^{N_\ell}\sum_{\substack{j=1\\j\neq i}}^{N_\ell}P_{\ell,k}^{(i)}P_{\ell,k}^{(j)},N_r\hat{P}_{r,z}\sum_{i=1}^{N_\ell}P_{\ell,z}^{(i)}\right]\\&=\frac{1}{N_\ell^3}\sum_{\substack{r=0\\r\neq\ell}}^{L}\frac{1}{N_r}\left(\sum_{i=1}^{N_\ell}\mathbb{C}ov\left[\left(P_{\ell,k}^{(i)}\right)^2,N_r\hat{P}_{r,z}P_{\ell,z}^{(i)}\right]+\mathbb{C}ov\left[N_r\hat{P}_{r,z}\sum_{i=1}^{N_\ell}P_{\ell,z}^{(i)},\sum_{i=1}^{N_\ell}\sum_{\substack{j=1\\j\neq i}}^{N_\ell}P_{\ell,k}^{(i)}P_{\ell,k}^{(j)}\right]\right)\\&=\frac{1}{N_\ell^3}\sum_{\substack{r=0\\r\neq\ell}}^{L}\frac{1}{N_r}\left(N_\ell\mathbb{C}ov\left[P_{\ell,k}^2,N_r\hat{P}_{r,z}P_{\ell,z}\right]+\mathbb{C}ov\left[N_r\hat{P}_{r,z}\sum_{i=1}^{N_\ell}P_{\ell,z}^{(i)},\sum_{i=1}^{N_\ell}\sum_{\substack{j=1\\j\neq i}}^{N_\ell}P_{\ell,k}^{(i)}P_{\ell,k}^{(j)}\right]\right).\end{aligned}\tag{B.5}$$

The first term of the previous expression, Eq. B.5, can be written as

$$\begin{aligned}\mathbb{C}ov\left[P_{\ell,k}^2,N_r\hat{P}_{r,z}P_{\ell,z}\right]&=\mathbb{C}ov\left[P_{\ell,k}^2,N_r\frac{1}{N_r}\sum_{j=1}^{N_r}P_{r,z}^{(j)}P_{\ell,z}\right]=N_r\mathbb{C}ov\left[P_{\ell,k}^2,P_{\ell,z}P_{r,z}\right]\\&=N_r\left(\mathbb{E}\left[P_{\ell,k}^2P_{\ell,z}\right]\mathbb{E}\left[P_{r,z}\right]-\mathbb{E}\left[P_{\ell,k}^2\right]\mathbb{E}\left[P_{\ell,z}\right]\mathbb{E}\left[P_{r,z}\right]\right)\end{aligned}\tag{B.6}$$

The second term of Eq. B.5 can be manipulated as it follows

$$
\begin{aligned}
&\mathbb{C}ov\left[N_r\hat{P}_{r,z}\sum_{i=1}^{N_\ell}P_{\ell,z}^{(i)},\sum_{i=1}^{N_\ell}\sum_{\substack{j=1\\j\neq i}}^{N_\ell}P_{\ell,k}^{(i)}P_{\ell,k}^{(j)}\right]\\
&=\mathbb{C}ov\left[N_r\hat{P}_{r,z}\sum_{i=1}^{N_\ell}P_{\ell,z}^{(i)},P_{\ell,k}^{(i)}\sum_{\substack{j=1\\j\neq i}}^{N_\ell}P_{\ell,k}^{(j)}+\sum_{\substack{q=1\\q\neq i}}^{N_\ell}P_{\ell,k}^{(q)}\left(P_{\ell,k}^{(i)}+\sum_{\substack{j=1\\j\neq q,i}}^{N_\ell}P_{\ell,k}^{(j)}\right)\right]\\
&=\mathbb{C}ov\left[N_r\hat{P}_{r,z}\sum_{i=1}^{N_\ell}P_{\ell,z}^{(i)},2P_{\ell,k}^{(i)}\sum_{\substack{j=1\\j\neq i}}^{N_\ell}P_{\ell,k}^{(j)}+\sum_{\substack{q=1\\q\neq i}}^{N_\ell}\sum_{\substack{j=1\\j\neq q,i}}^{N_\ell}P_{\ell,k}^{(q)}P_{\ell,k}^{(j)}\right]\\
&=2N_\ell\left(N_\ell-1\right)\mathbb{C}ov\left[N_r\hat{P}_{r,z}P_{\ell,z},P_{\ell,k}P_{\ell,k}'\right]\\
&=2N_\ell\left(N_\ell-1\right)\left(N_r\mathbb{E}\left[\hat{P}_{r,z}P_{\ell,z}P_{\ell,k}P_{\ell,k}'\right]-N_r\mathbb{E}\left[\hat{P}_{r,z}P_{\ell,z}\right]\mathbb{E}\left[P_{\ell,k}\right]^2\right)\\
&=2N_\ell\left(N_\ell-1\right)N_r\left(\mathbb{E}\left[P_{r,z}\right]\mathbb{E}\left[P_{\ell,z}P_{\ell,k}\right]\mathbb{E}\left[P_{\ell,k}\right]-\mathbb{E}\left[P_{r,z}\right]\mathbb{E}\left[P_{\ell,z}\right]\mathbb{E}\left[P_{\ell,k}\right]^2\right)
\end{aligned}
\tag{B.7}
$$

Finally, the last term of Eq. B.1 is written as

$$
\begin{aligned}
&\mathbb{C}ov\left[\sum_{\ell=0}^{L}\sum_{\substack{r=0\\r\neq\ell}}^{L}\hat{P}_{\ell,k}\hat{P}_{r,k},\sum_{\ell=0}^{L}\sum_{\substack{r=0\\r\neq\ell}}^{L}\hat{P}_{\ell,z}\hat{P}_{r,z}\right]\\
&=2\mathbb{C}ov\left[\sum_{\ell=0}^{L}\sum_{r=\ell+1}^{L}\hat{P}_{\ell,k}\hat{P}_{r,k},2\hat{P}_{\ell,z}\hat{P}_{r,z}+2\hat{P}_{\ell,z}\sum_{\substack{q=0\\q\neq\ell,r}}^{L}\hat{P}_{q,z}+2\hat{P}_{r,z}\sum_{\substack{q=0\\q\neq\ell,r}}^{L}\hat{P}_{q,z}+\sum_{\substack{q=0\\q\neq\ell,r}}^{L}\sum_{\substack{t=0\\t\neq q,\ell,r}}^{L}\hat{P}_{q,z}\hat{P}_{t,z}\right]\\
&=4\mathbb{C}ov\left[\sum_{\ell=0}^{L}\sum_{r=\ell+1}^{L}\hat{P}_{\ell,k}\hat{P}_{r,k},\hat{P}_{\ell,z}\hat{P}_{r,z}\right]+4\mathbb{C}ov\left[\sum_{\ell=0}^{L}\sum_{r=\ell+1}^{L}\hat{P}_{\ell,k}\hat{P}_{r,k},\hat{P}_{\ell,z}\sum_{\substack{q=0\\q\neq\ell,r}}^{L}\hat{P}_{q,z}\right]\\
&+4\mathbb{C}ov\left[\sum_{\ell=0}^{L}\sum_{r=\ell+1}^{L}\hat{P}_{\ell,k}\hat{P}_{r,k},\hat{P}_{r,z}\sum_{\substack{q=0\\q\neq\ell,r}}^{L}\hat{P}_{q,z}\right],
\end{aligned}
\tag{B.8}
$$

by using the single level derivation.

The first term of the previous expression can be simplified as

$$
\begin{aligned}
&\mathbb{C}ov\left[\sum_{\ell=0}^{L}\sum_{r=\ell+1}^{L}\hat{P}_{\ell,k}\hat{P}_{r,k},\hat{P}_{\ell,z}\hat{P}_{r,z}\right]=\sum_{\ell=0}^{L}\sum_{r=\ell+1}^{L}\mathbb{C}ov\left[\hat{P}_{\ell,k}\hat{P}_{r,k},\hat{P}_{\ell,z}\hat{P}_{r,z}\right]\\
&=\sum_{\ell=0}^{L}\sum_{r=\ell+1}^{L}\mathbb{C}ov\left[\left(\frac{1}{N_\ell}\sum_{i=1}^{N_\ell}P_{\ell,k}^{(i)}\right)\left(\frac{1}{N_r}\sum_{j=1}^{N_r}P_{r,k}^{(j)}\right),\left(\frac{1}{N_\ell}\sum_{i=1}^{N_\ell}P_{\ell,z}^{(i)}\right)\left(\frac{1}{N_r}\sum_{j=1}^{N_r}P_{r,z}^{(j)}\right)\right]\\
&=\sum_{\ell=0}^{L}\sum_{r=\ell+1}^{L}\frac{1}{N_\ell^2N_r^2}\mathbb{C}ov\left[\sum_{i=1}^{N_\ell}\sum_{j=1}^{N_r}P_{\ell,k}^{(i)}P_{r,k}^{(j)},P_{\ell,z}^{(i)}P_{r,z}^{(j)}+P_{\ell,z}^{(i)}\sum_{\substack{q=1\\q\neq j}}^{N_r}P_{r,z}^{(q)}+P_{r,z}^{(j)}\sum_{\substack{q=1\\q\neq i}}^{N_\ell}P_{\ell,z}^{(q)}+\sum_{\substack{q=1\\q\neq i}}^{N_\ell}P_{\ell,z}^{(q)}\sum_{\substack{t=1\\t\neq j}}^{N_r}P_{r,z}^{(t)}\right]\\
&=\sum_{\ell=0}^{L}\sum_{r=\ell+1}^{L}\frac{1}{N_\ell^2N_r^2}\left(N_\ell N_r\mathbb{C}ov\left[P_{\ell,k}P_{r,k},P_{\ell,z}P_{r,z}\right]+\left(N_r-1\right)N_\ell N_r\mathbb{C}ov\left[P_{\ell,k}P_{r,k},P_{\ell,z}P'_{r,z}\right]\right.\\
&\left.+\left(N_\ell-1\right)N_\ell N_r\mathbb{C}ov\left[P_{\ell,k}P_{r,k},P_{r,z}P'_{\ell,z}\right]\right)\\
&=\sum_{\ell=0}^{L}\sum_{r=\ell+1}^{L}\frac{1}{N_\ell N_r}\left(\mathbb{E}\left[P_{\ell,k}P_{\ell,z}\right]\mathbb{E}\left[P_{r,k}P_{r,z}\right]-\mathbb{E}\left[P_{\ell,k}\right]\mathbb{E}\left[P_{\ell,z}\right]\mathbb{E}\left[P_{r,k}\right]\mathbb{E}\left[P_{r,z}\right]\right.\\
&+\left(N_r-1\right)\mathbb{E}\left[P_{\ell,k}P_{\ell,z}\right]\mathbb{E}\left[P_{r,k}\right]\mathbb{E}\left[P_{r,z}\right]-\left(N_r-1\right)\mathbb{E}\left[P_{\ell,k}\right]\mathbb{E}\left[P_{\ell,z}\right]\mathbb{E}\left[P_{r,k}\right]\mathbb{E}\left[P_{r,z}\right]\\
&\left.+\left(N_\ell-1\right)\mathbb{E}\left[P_{\ell,k}\right]\mathbb{E}\left[P_{\ell,z}\right]\mathbb{E}\left[P_{r,k}P_{r,z}\right]-\left(N_\ell-1\right)\mathbb{E}\left[P_{\ell,k}\right]\mathbb{E}\left[P_{r,k}\right]\mathbb{E}\left[P_{r,z}\right]\mathbb{E}\left[P_{\ell,z}\right]\right)
\end{aligned}
\tag{B.9}
$$

The last two terms of Eq. B.8 are similar and can be obtained as demonstrated below for the first of them

$$
\begin{aligned}
&\mathbb{C}ov\left[\sum_{\ell=0}^{L}\sum_{r=\ell+1}^{L}\hat{P}_{\ell,k}\hat{P}_{r,k},\hat{P}_{\ell,z}\sum_{\substack{q=0\\q\neq\ell,r}}^{L}\hat{P}_{q,z}\right]=\sum_{\ell=0}^{L}\sum_{r=\ell+1}^{L}\mathbb{C}ov\left[\hat{P}_{\ell,k}\hat{P}_{r,k},\hat{P}_{\ell,z}\sum_{\substack{q=0\\q\neq\ell,r}}^{L}\hat{P}_{q,z}\right]\\
&=\sum_{\ell=0}^{L}\sum_{r=\ell+1}^{L}\mathbb{C}ov\left[\frac{1}{N_\ell}\frac{1}{N_r}\left(\sum_{i=1}^{N_\ell}P_{\ell,k}^{(i)}\right)\left(\sum_{j=1}^{N_r}P_{r,k}^{(j)}\right),\frac{1}{N_\ell}\frac{1}{N_q}\sum_{i=1}^{N_\ell}P_{\ell,z}^{(i)}\sum_{\substack{q=0\\q\neq\ell,r}}^{L}\left(\sum_{s=1}^{N_q}P_{q,z}^{(s)}\right)\right]\\
&=\sum_{\ell=0}^{L}\sum_{r=\ell+1}^{L}\frac{1}{N_\ell^2N_r}\mathbb{C}ov\left[\sum_{i=1}^{N_\ell}\sum_{j=1}^{N_r}P_{\ell,k}^{(i)}P_{r,k}^{(j)},\sum_{i=1}^{N_\ell}P_{\ell,z}^{(i)}\sum_{\substack{q=0\\q\neq\ell,r}}^{L}\frac{1}{N_q}\left(\sum_{s=1}^{N_q}P_{q,z}^{(s)}\right)\right]\\
&=\sum_{\ell=0}^{L}\sum_{r=\ell+1}^{L}\frac{1}{N_\ell^2N_r}\mathbb{C}ov\left[P_{\ell,k}^{(i)}\sum_{j=1}^{N_r}P_{r,k}^{(j)}+\sum_{\substack{q=1\\q\neq i}}^{N_\ell}\sum_{j=1}^{N_r}P_{\ell,k}^{(q)}P_{r,k}^{(j)},\sum_{i=1}^{N_\ell}P_{\ell,z}^{(i)}\sum_{\substack{q=0\\q\neq\ell,r}}^{L}\frac{1}{N_q}\left(\sum_{s=1}^{N_q}P_{q,z}^{(s)}\right)\right]\\
&=\sum_{\ell=0}^{L}\sum_{r=\ell+1}^{L}\frac{1}{N_\ell N_r}\mathbb{C}ov\left[P_{\ell,k}\sum_{j=1}^{N_r}P_{r,k}^{(j)},P_{\ell,z}\sum_{\substack{q=0\\q\neq\ell,r}}^{L}\frac{1}{N_q}\left(\sum_{s=1}^{N_q}P_{q,z}^{(s)}\right)\right]\\
&=\sum_{\ell=0}^{L}\frac{1}{N_\ell}\sum_{r=\ell+1}^{L}\sum_{\substack{q=0\\q\neq\ell,r}}^{L}\left(\mathbb{E}\left[P_{\ell,k}P_{\ell,z}\right]\mathbb{E}\left[P_{r,k}\right]\mathbb{E}\left[P_{q,z}\right]-\mathbb{E}\left[P_{\ell,k}\right]\mathbb{E}\left[P_{\ell,z}\right]\mathbb{E}\left[P_{r,k}\right]\mathbb{E}\left[P_{q,z}\right]\right)
\end{aligned}
\tag{B.10}
$$

□

TESTING THE LIMITATIONS OF THE NONVARIATIONAL FINITE ELEMENT METHOD FOR ELLIPTIC PDES

DIANA M. MORALES* AND KELSEY L. DIPIETRO†

Abstract. The optimal transport problem focuses on finding the optimal transportation path from one location to another with minimal cost. Its mathematical formulation leads to a PDE constrained optimization problem. Under the assumption of quadratic cost, the solution to the optimal transport problem is equivalent to solving the Monge-Ampère equation – a fully nonlinear second-order elliptic PDE. The numerical solution to the Monge-Ampère equation has recently been utilized to provide robust moving mesh adaptivity for solving partial differential equations. Because of its use for mesh adaptivity and several other applications, its numerical solution has been the recent topic of much research, including finite difference, biharmonic finite element, and least squares methods. This work focuses on implementing a finite element formulation of the Monge-Ampère equation, specifically in nonvariational form, following the work of [18, 19], using Intrelab, a subdirectory of the Trilinos package Intrepid [16] that provides an interface between Intrepid and Matlab. We present convergence studies of using the nonvariational finite element method for elliptic problems that cannot be put into the standard variational form. We show and present results on applying the nonvariational finite element methods to fully nonlinear elliptic problems, with a particular focus on the Monge-Ampère equation.

1. Introduction. Optimal mass transport theory has a wide range of applications. Application examples include data science for image processing, mesh adaptivity, machine learning, and seismic imaging [20, 2, 14, 9, 11, 6, 10, 13, 2, 23]. The optimal transport problem focuses on finding the optimal transportation path of a density from one location to another with minimal cost, as shown in Figure 1.1.

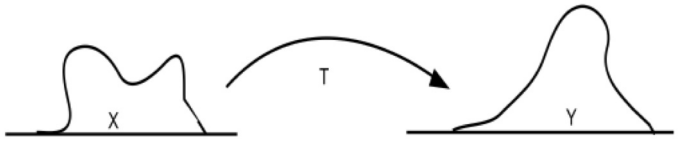


Fig. 1.1: An illustration of optimal transportation map: $T : X \to Y$ [24]

This problem was originally posed by Gaspard Monge in 1781 [5]. Mathematician and economist Leonid Kantorovich further progressed the transportation theory field during World War II. For this reason, the problem formulation is often times referred to as the Monge-Kantorovich (MK) transportation problem. The Kantorovich formulation of the transportation problem can be seen as a generalization of the Monge formulation. Monge's original optimal transport problem is as follows: given $\mu \in \mathcal{P}(X)$ and $\nu \in \mathcal{P}(Y)$,

$$\text{minimize } M(T) = \int_X c(x, T(x)) d\mu(x) \tag{1.1}$$

subject to $\nu = T_*\mu$ and over μ-measurable maps $T : X \to Y$.

Under quadratic cost $c(\mathbf{x}) = \frac{1}{2}|T(x) - x|^2$, Brenier [4] showed that the solution of the optimal transport problem is the gradient of a convex mesh potential, such that $T(\mathbf{x}) = \nabla u$.

*University of Notre Dame, dmorale3@nd.edu
†Sandia National Laboratories, kdipiet@sandia.gov

Using this relationship and a change of variables equation (1.1) simplifies to finding the solution of the Monge-Ampère equation,

$$\det(D^2u) = f, \text{ in } \Omega, \quad u = 0 \text{ on } \partial\Omega, \tag{1.2}$$

where D^2u is the Hessian matrix of u.

The Monge-Ampère equation is a fully nonlinear elliptic second-order PDE. Monge-Ampère equations were first studied by Gaspard Monge, in 1784, followed by André-Marie Ampère in 1820. The numerical solution to the Monge-Ampère equation can be found through varying methods, such as finite difference [9, 3, 14, 21], finite element methods [12, 20], least squares method [22], and by directly solving the optimization form [15]. The methods range in success, some can only find weak or viscosity solutions and others suffer from instabilities due to the conditional ellipticity of the operators. We do not summarize the benefits and disadvantages of each of the methods in this article, but point the interested reader to the review article [7]. Because of its success of creating adaptive meshes for partial differential equations [20], which is the ultimate goal of this research, the numerical method utilized in this article is the nonvariational finite element method of Lakkis and Pryer [18, 17].

2. Nonvariational Finite Element Method. The mathematical formulation of the nonvariational finite element method [NVFEM] described in [18] begins by finding u such that

$$\mathbf{A} : D^2u = f \text{ in } \Omega \text{ and } u|_{\partial\Omega} = g. \tag{2.1}$$

Following standard FEM notation, $\mathbf{A} : \Omega \to \mathbb{R}^{d\times d}$ is the coefficient matrix. For this type of second order elliptic boundary value problem, the standard finite element method might not always yield the appropriate solution. The standard FEM results in rewriting the operator in divergence form which can potentially introduce a convective term. This leaves open the possibility of the problem becoming convection-dominated and unstable for FEM. Instead, [18] suggests directly discretizing the strong form over the standard weak form discretization of FEM. For this method, an appropriate finite element Hessian approximation for the elliptic operator is needed. Introducing the following finite element spaces:

$$\mathbb{V} := \{\Phi \in H^1(\Omega) : \Phi|_K \in \mathbb{P}^p, \forall K \in \tau\} \tag{2.2a}$$

$$\mathring{\mathbb{V}} := \mathbb{V} \cap H_0^1(\Omega) = \{\Phi \in \mathbb{V} : \Phi|_{\partial\Omega} = 0\}, \tag{2.2b}$$

where $\mathbb{P}^k$ is the linear space of polynomials in d variables of degrees no higher than $k > 0$. The mesh size function is denoted as $h(\mathbf{x})$. The problem is integrated against the test function $\phi \in H_0^1(\Omega)$ as

$$\langle \mathcal{L}u, \phi\rangle = \langle \mathbf{A} : D^2u, \phi\rangle = \langle f, \phi\rangle. \tag{2.3}$$

In order to discretize (2.3), [18] find a finite element approximation of the Hessian. The distribution of the Hessian is typically given as a function $v \in H^1(\Omega)$ defined as

$$\langle D^2v|\phi\rangle = -\langle \nabla v \otimes \nabla\phi\rangle \quad \forall\phi \in C_0^\infty(\Omega), \tag{2.4}$$

where $C_0^\infty(\Omega)$ denotes the Schwartz class of functions on Ω given as,

$$C_0^\infty(\Omega) := \{\phi \in C^\infty(\Omega) : \text{supp } \phi \text{ compact in } \Omega\}. \tag{2.5}$$

An important thing to note is that ϕ cannot be chosen as $\phi \in \mathring{\mathbb{V}}$ since too much information will be lost on the boundary and the finite element Hessian will not necessarily be zero on the boundary. For this reason, the test function ϕ must be specialized. The domain of the function $D^2 v$ in (2.4) must be extended so that it includes some test functions that aren't compactly supported. An extension can be found by letting $\mathbf{n} : \partial\Omega \to \mathbb{R}^d$ be the outward pointing normal of Ω, taking $v \in C^2(\Omega) \cap C^1(\overline{\Omega})$, and using integration by parts to get

$$\langle D^2 v, \phi \rangle = -\langle \nabla v \otimes \nabla \phi \rangle + \langle \nabla v \otimes \mathbf{n} \phi \rangle_{\partial\Omega} \quad \forall\, \phi \in C^1(\Omega) \cap C^0(\overline{\Omega}). \tag{2.6}$$

We define $v \in \mathbb{V}$ as a piecewise polynomial in the triangulation. We know it is continuous, but potentially not differentiable. For this reason, the gradient ∇v is a function in $\mathbb{P}^{p-1}$ and the limit at the boundary is defined almost everywhere. Note the right hand side for (2.6) is well defined for $\phi \in \mathbb{V}$ and is equivalent to the right hand side of (2.4) when $\phi = 0$ for boundary values. This results in the following finite element form of the Hessian of $V \in \mathring{\mathbb{V}}$ as the unique $\mathbf{H}[V] \in \mathbb{V}$ such that

$$\langle \mathbf{H}[V], \Phi \rangle := -\langle \nabla V \otimes \nabla \Phi \rangle + \langle \nabla V \otimes \mathbf{n} \Phi \rangle_{\partial\Omega} \quad \forall\, \Phi \in \mathbb{V}, \tag{2.7}$$

implying $\mathbf{H}$ is a linear operator on $\mathring{\mathbb{V}}$. Inserting the Hessian into the elliptic model problem reduces the problem to finding $U \in \mathring{\mathbb{V}}$ such that

$$\langle \mathbf{A} : \mathbf{H}[U], \mathring{\Phi} \rangle = \langle f, \mathring{\Phi} \rangle \quad \forall\, \mathring{\Phi} \in \mathring{\mathbb{V}} \tag{2.8}$$

The final nonvariational finite element form for the discretization of (2.8) is given as $U = \mathring{\mathbf{\Phi}}\mathbf{u}$, where $\mathbf{u} \in \mathbb{R}^N$ is the solution to the linear system:

$$\mathbf{D}\mathbf{u} := \sum_{\alpha=1}^{d} \sum_{\beta=1}^{d} \mathbf{B}^{\alpha,\beta} \mathbf{M}^{-1} \mathbf{C}_{\alpha,\beta} \mathbf{u} = \mathbf{f} \tag{2.9}$$

with the following components of (2.9):

$$\mathbf{B}^{\alpha,\beta} := \langle \mathring{\Phi}, \mathbf{A}^{\alpha,\beta} \Phi^T \rangle \in \mathbb{R}^{\mathring{N} \times N} \tag{2.10a}$$

$$\mathbf{M} := \langle \Phi, \Phi^T \rangle \in \mathbb{R}^{N \times N} \tag{2.10b}$$

$$\mathbf{C}_{\alpha,\beta} := -\langle \partial_\beta \Phi, \partial_\alpha \mathring{\Phi}^T \rangle + \langle \Phi n_\beta, \partial_\alpha \mathring{\Phi}^T \rangle_{\partial\Omega} \in \mathbb{R}^{N \times \mathring{N}} \tag{2.10c}$$

$$\mathbf{f} := \langle f, \mathring{\Phi} \rangle \in \mathbb{R}^{\mathring{N}} \tag{2.10d}$$

This formulation is generally not sparse, making several efficient iterative methods essentially useless to finding a solution. In order to recover sparsity in the formulation, [18] suggest augmenting the solution space with the Hessian by using a generalized Schur complement. The matrix $\mathbf{D}$ is the sum of Schur complements $\mathbf{B}^{\alpha,\beta}\ \mathbf{M}^{-1}\ \mathbf{C}_{\alpha,\beta}$, therefore a new $(d^2+1)^2$ block matrix $\mathbf{E}$ can be introduced (letting $d = 2$ in this case),

$$\mathbf{E} = \begin{bmatrix} \mathbf{M} & \mathbf{0} & \mathbf{0} & \mathbf{0} & -\mathbf{C_{1,1}} \\ \mathbf{0} & \mathbf{M} & \mathbf{0} & \mathbf{0} & -\mathbf{C_{1,2}} \\ \mathbf{0} & \mathbf{0} & \mathbf{M} & \mathbf{0} & -\mathbf{C_{2,1}} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{M} & -\mathbf{C_{2,2}} \\ \mathbf{B^{1,1}} & \mathbf{B^{1,2}} & \mathbf{B^{2,1}} & \mathbf{B^{2,2}} & \mathbf{0} \end{bmatrix} \tag{2.11}$$

It has been shown in [18] that solving the system

$$\mathbf{E}\mathbf{v} = \mathbf{b}, \tag{2.12}$$

with $\mathbf{v} = (\mathbf{h}_{1,1}, \mathbf{h}_{1,2}, \mathbf{h}_{2,1}, \mathbf{h}_{2,2}, \mathbf{u})^T$ and $\mathbf{b} = (\mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{f})^T$ is equivalent to solving the system (2.9). This will be the structure we use to solve all our examples, including the nonlinear cases given in the following sections.

2.1. Nonhomogeneous Dirichlet Boundary Conditions. In general, the elliptic problems we want to solve, particularly the Monge-Ampère equation, do not have homogeneous Dirichlet boundary conditions. For now, we just address nonhomogeneous Dirichlet conditions and leave the application of other physically relevant boundary conditions for future work.

Under the assumption of Dirichlet boundary conditions $u = g \,\text{on}\, \partial\Omega$, we can enforce them through the block representation (where **dc** stands for dirichlet condition boundary nodes),

$$\begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{E_{dc}} & \mathbf{E} \end{bmatrix} \begin{bmatrix} \mathbf{v_{dc}} \\ \mathbf{v} \end{bmatrix} = \begin{bmatrix} \mathbf{b_{dc}} \\ \mathbf{b} \end{bmatrix}. \tag{2.13}$$

Where $\mathbf{v_{dc}} = [\mathbf{h}^{\mathbf{dc}}_{1,1}, \mathbf{h}^{\mathbf{dc}}_{1,2}, \mathbf{h}^{\mathbf{dc}}_{2,1}, \mathbf{h}^{\mathbf{dc}}_{2,2}, \mathbf{u^{dc}}]^T, \mathbf{b_{dc}} = [\mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{g}]^T$. The matrix $\mathbf{E_{dc}}$ is defined as

$$\mathbf{E} = \begin{bmatrix} \mathbf{M} & \mathbf{0} & \mathbf{0} & \mathbf{0} & -\mathbf{C}^{\mathbf{dc}}_{1,1} \\ \mathbf{0} & \mathbf{M} & \mathbf{0} & \mathbf{0} & -\mathbf{C}^{\mathbf{dc}}_{1,2} \\ \mathbf{0} & \mathbf{0} & \mathbf{M} & \mathbf{0} & -\mathbf{C}^{\mathbf{dc}}_{2,1} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{M} & -\mathbf{C}^{\mathbf{dc}}_{2,2} \\ \mathbf{B}^{1,1} & \mathbf{B}^{1,2} & \mathbf{B}^{2,1} & \mathbf{B}^{2,2} & \mathbf{0}. \end{bmatrix}. \tag{2.14}$$

Given a $\boldsymbol{\Phi}_{\mathbf{dc}} = \{\Phi_1, \ldots \Phi_{N_{\mathbf{dc}}}\}$, the Dirchlet components of (2.14) can be defined as

$$\mathbf{C}^{\mathbf{dc}}_{\alpha,\beta} = - < \partial_\beta \boldsymbol{\Phi}, \partial_\alpha \boldsymbol{\Phi}^T_{\mathbf{dc}} > + < \boldsymbol{\Phi}\mathbf{n}_\beta, \partial_\alpha \boldsymbol{\Phi}^T_{\mathbf{dc}} >_{\partial\Omega} \in \mathbb{R}^{N \times N_{\mathbf{dc}}}, \tag{2.15a}$$

$$\mathbf{g}_j = g(x_j)\Phi_j \in \mathbb{R}^{N_{\mathbf{dc}}} \tag{2.15b}$$

The block matrix (2.13) is trivially solved as

$$\mathbf{Ev} = \mathbf{b} - \mathbf{E_{dc}}\mathbf{b_{dc}}. \tag{2.16}$$

Sections 3.3 and 3.4 gives an example of a nonhomogenous boundary condition applied using this method.

3. Examples of the Nonvariational Finite Element Method. Since the ultimate goal of our project is to create robust solution methods for the Monge-Ampère equation in the Trilinos [25], we begin by recreating the test problems from [18] using Intrelab and Intrepid [16]. In [18], four benchmark problems are introduced. The following experiments are taken on Ω, where Ω is the square $S = [-1, 1] \times [-1, 1] \subset \mathbb{R}^2$. For the first two examples, we let the diffusion matrix be

$$\mathbf{A}(\mathbf{x}) = \begin{bmatrix} 1 & b(\mathbf{x}) \\ b(\mathbf{x}) & a(\mathbf{x}) \end{bmatrix} \tag{3.1}$$

3.1. Nondifferentiable Coefficient. Nondifferentiable coefficients are typically challenging for standard finite element methods. This example will demonstrate how the nonvariational finite element method will fair with these coefficients. We take $a(\mathbf{x})$ and $b(\mathbf{x})$ to be the following:

$$\begin{aligned} a(\mathbf{x}) &= \left(x_1^2 x_2^2\right)^{1/3} + 1, \\ b(\mathbf{x}) &= 0 \end{aligned} \tag{3.2}$$

and choose the right hand side f to be such that the exact solution is $u(\mathbf{x}) = \exp\left(-10|\mathbf{x}|^2\right)$. Figure 3.1 gives a comparison of the computed solution of 2.12 discretized with linear $\mathbb{P}^1$ finite elements using Intrelab in MATLAB on a 64×64 grid, where the system is solved with either the backslash operator or an LU-decomposition. Our solution follows the expected order of convergence for $\mathbb{P}^1$ elements, as seen in Figure 3.2. This shows that our Intrelab implementation is working as expected and confirms the results of Lakkis and Pryer [18].

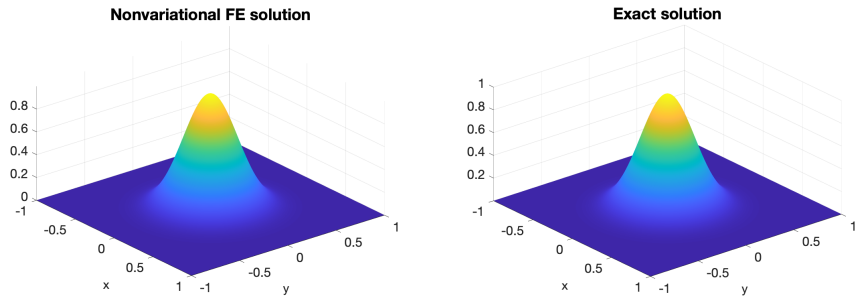


Fig. 3.1: The comparison between the Nonvariatonal finite element method (left) and the exact solution (right) for example 3.1 on a 64×64 grid

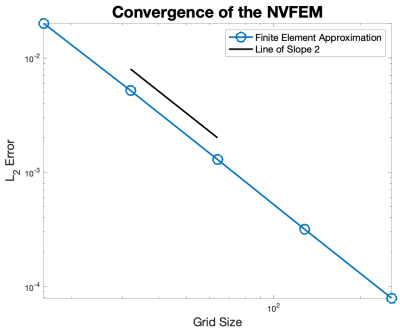


Fig. 3.2: Second order convergence for the Nondifferentiable operator example 3.1

3.2. Convection Dominated Coefficient. For this experiment, we take a convection dominated operator. This example demonstrates how the nonvariational numerical method can avoid some of the undesired effects of the standard finite element method when dealing with convection dominated operators. We take matrix $\mathbf{A}$ in (3.1) with

$$\begin{aligned} a(\mathbf{x}) &= \arctan\left(K(|\mathbf{x}|^2 - 1)\right) + 2, \\ b(\mathbf{x}) &= 0 \end{aligned} \tag{3.3}$$

The standard finite element method, written in divergence form, would oscillate for large values of K. We choose the right hand side f to be such that the exact solution to the example is $u(\mathbf{x}) = \sin(\pi x_1)\sin(\pi x_2)$. It's shown in 3.3 that the NVFEM does not result in the oscillations expected with the standard finite element method.

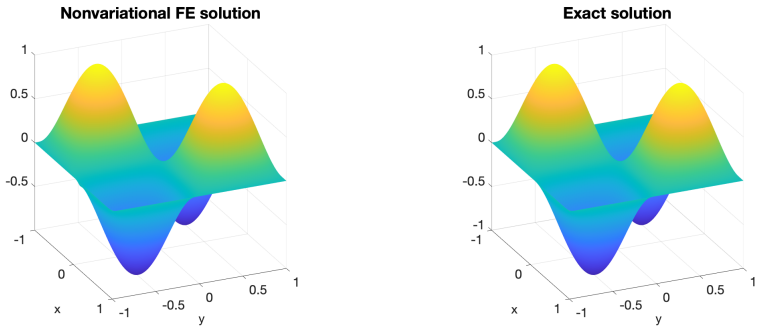


Fig. 3.3: The comparison between the Nonvariatonal finite element method (left) and the exact solution (right) for example 3.2 on a 64×64 grid

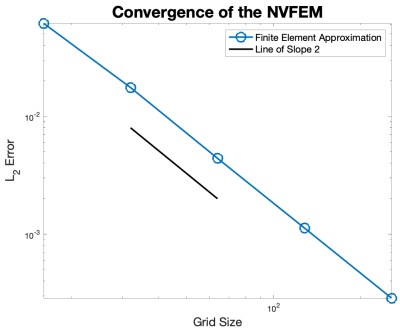


Fig. 3.4: Second order convergence for the convection dominated coefficient example 3.2

3.3. Singular Solution. The following example is to test a singular solution. In this case, we let $a(\mathbf{x})$ and $b(\mathbf{x})$ in 3.1 be:

$$\begin{aligned} a(\mathbf{x}) &= \sin\left(\frac{1}{|x_1| + |x_2| + 10^{-15}}\right) + 2, \\ b(\mathbf{x}) &= 0 \end{aligned} \tag{3.4}$$

This operator in particular has oscillations at the origin, at $\mathbf{0}$. We let f be such that the exact solution is $u(\mathbf{x}) = \left(2 - x_1^2 - x_2^2\right)^{1/2}$. This solution $u \in H^1(\Omega)$ but $u \notin H^2(\Omega)$. The domain has singularities on the corners and therefore a quadratic convergence rate is not achieved.

3.4. Nonsymmetric Operator. For this example, the operator is chosen so that $b(\mathbf{x})$ is nonzero. Ellipticity must still be maintained, therefore $a(\mathbf{x})$ is chosen such that the trace of $\mathbf{A}$ overtakes its determinant. We let $a(\mathbf{x})$ and $b(\mathbf{x})$ be the following:

$$\begin{aligned} a(\mathbf{x}) &= 2, \\ b(\mathbf{x}) &= \left(x_1^2 x_2^2\right)^{1/3} \end{aligned} \tag{3.5}$$

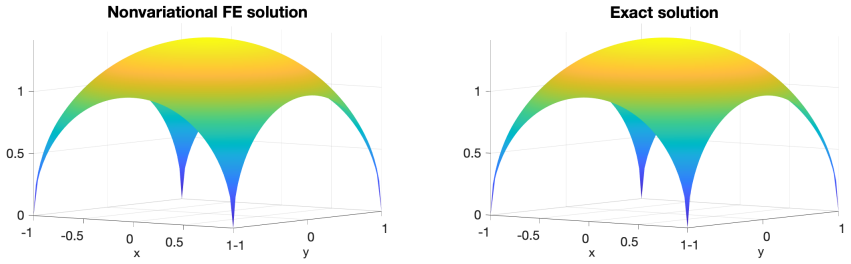


Fig. 3.5: The comparison between the Nonvariatonal finite element method (left) and the exact solution (right) for example 3.3 on a 64×64 grid

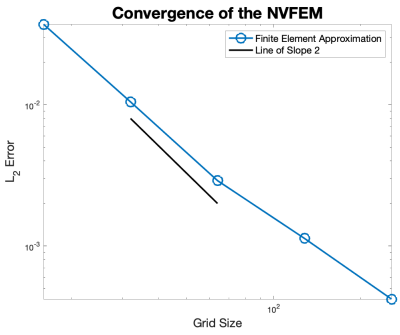


Fig. 3.6: Convergence for the singular solution example 3.3

We also let the f right hand side be chosen such that the exact solution is:

$$u(\mathbf{x}) = \begin{cases} \frac{x_1 x_2 \left(x_1^2 - x_2^2\right)}{x_1^2 + x_2^2}, & \mathbf{x} \neq \mathbf{0} \\ 0, & \mathbf{x} = \mathbf{0} \end{cases} \tag{3.6}$$

This creates a nonsymmetric Hessian operator. Figure 3.7 shows the comparison between the nonvariational method and the exact solution while Figure 3.8 shows the convergence graph of the nonvariational method.

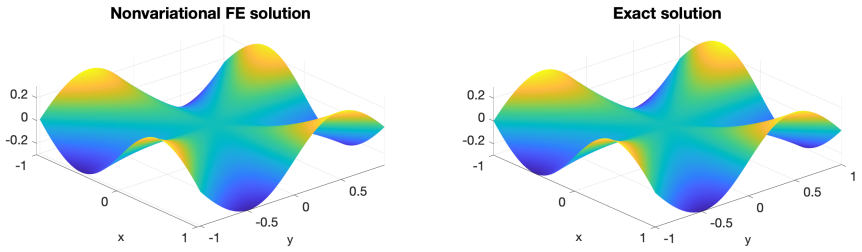


Fig. 3.7: The comparison between the Nonvariatonal finite element method (left) and the exact solution (right) for example 3.4 on a 64×64 grid

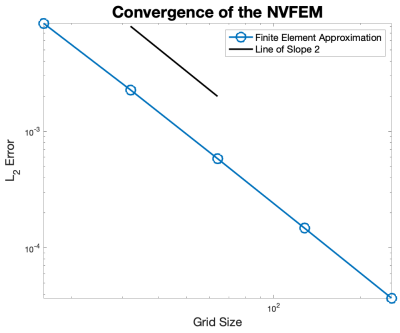


Fig. 3.8: Second order convergence for the Nonsymmetric operator example 3.4

4. Nonvariational Method for Nonlinear Elliptic Problems. Since our ultimate goal is to have an efficient finite element numerical solver for the Monge-Ampère equation for moving mesh adaptive methods for solving PDEs, we need to address how the nonvariational method described in Section 2 applies to nonlinear partial differential equations. Following the lead of [19], we can construct an iterative method to find the solution of the fully nonlinear PDE. Assuming a model problem,

$$\mathcal{N}[u] = F(D^2u) - f = 0, \tag{4.1}$$

where D^2u is the Hessian of u. The authors in [19] propose using a Newton's method to solve (4.1), though other nonlinear solvers could be considered. We focus on implementing the approach where the Newton's method is applied to (4.1) and the resulting Newton's step is discretized using the NVFEM. This equates to solving the following system

$$\mathbf{N}(D^2u^n) : D^2u^{n+1} = g(D^2u^n), \tag{4.2}$$

where

$$\mathbf{N}(\mathbf{X}) := F'(\mathbf{X}), \tag{4.3a}$$

$$g(\mathbf{X}) := f - F(\mathbf{X}) + F'(\mathbf{X}) : \mathbf{X}. \tag{4.3b}$$

Equation (4.3) can be seen as a form of (2.1) and can be discretized in finite element space using the Nonvariational finite element method. This can be stated as given an initial guess $U^0 := \Pi_0 u^0$ for each $n \in \mathbb{N}_0$, find $U^{n+1}, \mathbf{H}[U^{n+1}] \in \mathring{\mathbb{V}} \times \mathbb{V}^{d\times d}$ such that,

$$< \mathbf{H}[U^{n+1}], \Phi > + < \nabla U^{n+1} \otimes \nabla \Phi - < \nabla U^{n+1} \otimes \mathbf{n}\Phi >= \mathbf{0}, \quad \forall \Phi \in \mathbb{V} \tag{4.4a}$$

$$< \mathbf{N}(\mathbf{H}[U^n]) : \mathbf{H}[U^{n+1}], \Psi >=< g(\mathbf{H}[U^n], \Psi >, \quad \Psi \in \mathring{\mathbb{V}}, \tag{4.4b}$$

where $\mathbf{H}[U] = D^2U$, the discrete Hessian formulation (2.7). From equations (4.3),(4.4) we can now apply the method to a variety of nonlinear partial diffential equations, with a particular target on the Monge-Ampère equation.

4.1. Test Problem: Smooth Nonlinearity. To test our implementation of the method, we begin with a simple fully nonlinear problem with a smooth nonlinearity. The nonlinear problem considered is

$$\mathcal{N}[u] := \sin(\Delta u) + 2\Delta u - f = 0 \text{ in } \Omega \tag{4.5a}$$

$$u = 0 \text{ on } \partial\Omega \tag{4.5b}$$

Taking a Newton's step on (4.5) leads us to the problem of given an initial guess u^0, for $n \in \mathbb{N}_0$ find u^{n+1} such that

$$(\cos(\Delta u^n) + 2)\mathbf{I} : D^2(u^{n+1} - u^n) = f - \sin(\Delta u^n) - 2\Delta u^n. \tag{4.6}$$

Which can be easily discretized using (4.4). The example is calculated in 2D on a square domain $\Omega = [-1, 1]^2$ and triangular mesh. The right hand side of (4.5) is chosen so that the exact solution $u(\mathbf{x}) = \exp\left(-10|\mathbf{x}|^2\right)$. The initial guess is chosen to be $u^0, H^0 = 0$.

Figure 4.1 shows a comparison between the computed solution using the NVFEM and the exact solution. We observe that the method attains optimal convergence for $\mathbb{P}^1$ finite elements and can be solved to residual tolerance of 10^{-12}. This problem took 8 iterations to converge and is a critical stepping stone into formulating the Monge-Amère equation to the NVFEM using Intrelab.

4.2. Test Problem: The Monge-Ampère Equation. The main reason we have focused on the nonvariational finite element method is that it can be used to solve the Monge-Ampère equation for adaptive meshing algorithms. In this section, we outline how the method can be applied to the Monge-Ampère equation.

In particular, Lakkis and Pryer note in [19], that there are certain limitations in applying the methods of Section 4 to the Monge-Ampère equation. The equation (1.2) has conditional ellipticity, which can cause convergence issues if initial guess is not carefully chosen [18, 1]. Therefore, we must make sure that the initial Newton's guess u^0 is strictly convex. Also, special care must be taken to ensure of the convexity of the subsequent Newton iterations [18].

Recalling the form of the Monge-Ampère equation (1.2) $\det(D^2u) = f$, we can put in into the Newton's form outlined in 4.

To impose necessary convexity conditions the Newton's method for (1.2) can then be given as

$$D\mathcal{N}[u]v = \mathrm{Cof}D^2u : D^2v, \tag{4.7a}$$

$$\mathbf{N}(D^2u^n) = \mathrm{Cof}D^2u^n, \tag{4.7b}$$

$$g(D^2u^n) = f - \det D^2u^n + \mathrm{Cof}D^2u^n : D^2u^n. \tag{4.7c}$$

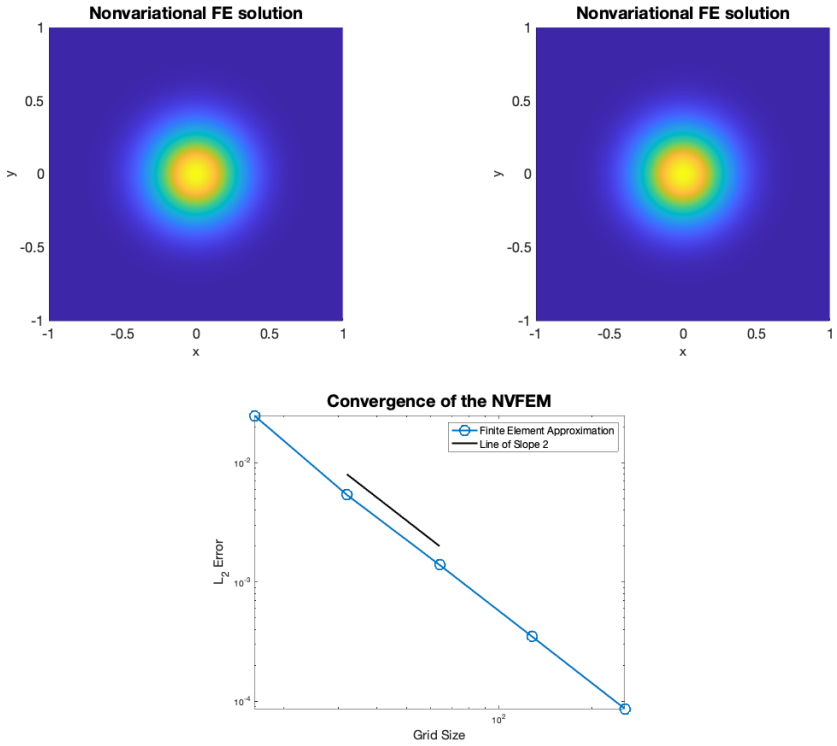


Fig. 4.1: The comparison between the Nonvariatonal finite element method (left) and the exact solution (middle) for example (4.5) on a 64×64 grid until convergence is reached ($\text{tol} = 1e-12$). The right figure gives the expected quadratic convergence for the solution as the grid is refined.

Discretizing (4.7) becomes, for an initial guess $U^0 := \Pi_0 u^0$ for each $n \in \mathbb{N}_0$ find $(U^{n+1}, \mathbf{H}[U^{n+1}]) \in \overset{\circ}{\mathbb{V}} \times \mathbb{V}^{d\times d}$

$$< \mathbf{H}[U^{n+1}], \Phi > + \int_\Omega \nabla U^{n+1} \otimes \Phi - \int_{\partial\Omega} \nabla U^{n+1} \otimes \mathbf{n}\Phi = \mathbf{0} \quad \forall \Phi \in \mathbb{V} \tag{4.8a}$$

$$< \text{Cof} D^2 U^n : \mathbf{H}[U^{n+1}], \Psi >=< f + \det D^2 U^n, \Psi > \ \forall \Psi \in \overset{\circ}{\mathbb{V}}. \tag{4.8b}$$

Where $\mathbf{Cof}(D^2U^n)$ is the cofactor matrix of the Hessian.

This falls into the same form as the nonvariational finite element method (2.1), where the coefficient matrix A has been replaced by the cofactor matrix of the Hessian, given by the solution. In addition, the right hand side as additional dependence on the determinant of the Hessian D^2U^N, which once again should be easy to calculate from the solution, given that we calculate D^2U^N. Since we will be solving the expanded system for both D^2U^N, U^N, we should be able to calculate the cofactor matrix from the Hessian matrix quite easily from the previous solution of D^2U^N. That allows us to calculate both D^2U^{N+1}, U^{N+1} iteratively until the Newton's method converges.

A big challenge of using this method is coming up with an initial guess both for U

and the Hessian $\mathbf{H}[U^0]$. Due to ellipticity constraints depending on the convexity of the problem, we want the initial guess to be convex. In addition, it is preferable to not have to interpolate the Hessian values from U^0, but rather compute them directly from the initial condition. As the initial condition for the Newton's step and nonvariational finite element method (4.8a),(4.7), Lakkis and Pryer use a trick from Dean and Glowinski [8] and find the initial condition by solving the linear problem,

$$\Delta U^0 = 2\sqrt{f} \text{ in } \Omega \tag{4.9a}$$

$$U^0 = g \text{ on } \partial\Omega \tag{4.9b}$$

4.3. Monge-Ampere in Intrelab. Because of the conditional ellipticity of the Monge-Ampére, particular care needs to be taken to implement it using Intrelab. At the time of submission of this article, we have implemented a Newton's method with a NVFEM discretization of the Monge-Ampére equation. However, for a simple test problem, we observe that the method is not converging in Intrelab, likely due to a loss of convexity between subsequent Newton's iterations. We are investigating possible remedies for the convergence issues that include increasing the approximation order of the finite elements (would need to be migrated out of Intrelab, which only offers $\mathbb{P}^1$ elements), imposing convexity through constraints using PDE constrained optimization, and implementing other nonlinear solvers.

5. Conclusion. We have done a thorough analysis of Lakkis and Pryer's [18] NVFEM and completed an implementation of the four benchmark problems using Intrelab. This was a necessary step to determine if Intrelab could adequately handle the NVFEM. We demonstrated the desired results and convergence rates for the benchmark problems. Now that the framework for using the NVFEM with Intrelab is established, we will use both this method to solve the Monge-Ampère equation, with the ultimate goal of creating robust moving mesh adaptive methods for solving singular nonlinear PDEs.

REFERENCES

[1] N. E. AGUILERA AND P. MORIN, *On convex functions and the finite element method*, SIAM Journal on Numerical Analysis, 47 (2009), pp. 3139–3157.

[2] M. ARJOVSKY, S. CHINTALA, AND L. BOTTOU, *Wasserstein gan*, 2017.

[3] J.-D. BENAMOU, B. D. FROESE, AND A. M. OBERMAN, *Numerical solution of the optimal transportation problem using the monge-ampére equation*, Journal of Computational Physics, 260 (2014), pp. 107 – 126.

[4] Y. BRENIER, *Polar factorization and monotone rearrangement of vector-valued functions*, Communications on Pure and Applied Mathematics, 44 (1991), pp. 375–417.

[5] CAYLEY, *On monge's mémoire sur la théorie des déblais et des remblais*, Proceedings of the London Mathematical Society, s1-14 (1882), pp. 139–143.

[6] J. CHEN, Y. CHEN, H. WU, AND D. YANG, *The quadratic wasserstein metric for earthquake location*, Journal of Computational Physics, 373 (2018), pp. 188 – 209.

[7] E. DEAN AND R. GLOWINSKI, *Numerical methods for fully nonlinear elliptic equations of the monge-ampére type*, Computer Methods in Applied Mechanics and Engineering, 195 (2006), pp. 1344 – 1386. A Tribute to Thomas J.R. Hughes on the Occasion of his 60th Birthday.

[8] E. DEAN AND R. GLOWINSKI, *Numerical methods for fully nonlinear elliptic equations of the monge-ampére type*, Computer Methods in Applied Mechanics and Engineering, 195 (2006), pp. 1344–1386.

[9] K. L. DIPIETRO AND A. E. LINDSAY, *Adaptive solution to two-dimensional partial differential equations in curved domains using the monge-ampére equation*, SIAM Journal on Scientific Computing, 41 (2019), pp. A1331–A1356.

[10] Y. DUKLER, W. LI, A. LIN, AND G. MONTUFAR, *Wasserstein of Wasserstein loss for learning generative models*, in Proceedings of the 36th International Conference on Machine Learning, K. Chaudhuri and R. Salakhutdinov, eds., vol. 97 of Proceedings of Machine Learning Research, Long Beach, California, USA, 09–15 Jun 2019, PMLR, pp. 1716–1725.

[11] B. E. Engquist and Y. Yang, *Seismic imaging and optimal transport.*, 2018.
[12] X. Feng and M. Neilan, *Analysis of galerkin methods for the fully nonlinear monge-ampére equation*, Journal of Scientific Computing, 47 (2011), pp. 303–327.
[13] N. Feyeux, A. Vidard, and M. Nodet, *Optimal transport for variational data assimilation*, Nonlinear Processes in Geophysics, 25 (2018), pp. 55–66.
[14] B. D. Froese, *A numerical method for the elliptic monge-ampére equation with transport boundary conditions*, SIAM Journal on Scientific Computing, 34 (2012), pp. A1432–A1459.
[15] E. Haber and A. Tannenbaum, *An efficient numerical method for the solution of the l_2 optimal mass transfer problem*, SIAM Journal on Scientific Computing, 32 (2010), pp. 197–211.
[16] T. Intrepid Project Team, *The Intrepid Project Website*.
[17] E. Kawecki, O. Lakkis, and T. Pryer, *A finite element method for the monge-ampére equation with transport boundary conditions*, 2018.
[18] O. Lakkis and T. Pryer, *A finite element method for second order nonvariational elliptic problems*, SIAM Journal on Scientific Computing, 33 (2011), pp. 786–801.
[19] ———, *A finite element method for nonlinear elliptic problems*, SIAM Journal on Scientific Computing, 35 (2013), pp. A2025–A2045.
[20] A. T. T. McRae, C. J. Cotter, and C. J. Budd, *Optimal-transport–based mesh adaptivity on the plane and sphere using finite elements*, SIAM Journal on Scientific Computing, 40 (2018), pp. A1121–A1148.
[21] L. Métivier, R. Brossier, Q. Mérigot, E. Oudet, and J. Virieux, *An optimal transport approach for seismic tomography: application to 3d full waveform inversion*, Inverse Problems, 32 (2016), p. 115008.
[22] C. R. Prins, R. Beltman, J. H. M. ten Thije Boonkkamp, W. IJzerman, and T. W. Tukker, *A least-squares method for optimal transport using the monge-ampére equation*, SIAM Journal on Scientific Computing, 37 (2015), pp. B937–B961.
[23] T. Salimans, H. Zhang, A. Radford, and D. Metaxas, *Improving gans using optimal transport*, 2018.
[24] G. Savarè, *Optimal Transport*, 2015 (accessed August 14, 2020).
[25] T. Trilinos Project Team, *The Trilinos Project Website*, 2020 (acccessed August 26, 2020).

PARAMETER SENSITIVITY ANALYSIS OF THE SPARTEN HIGH PERFORMANCE SPARSE TENSOR DECOMPOSITION SOFTWARE

JEREMY M. MYERS*, DANIEL M. DUNLAVY†, KEITA TERANISHI‡, AND D.S. HOLLMAN§

Abstract. Tensor decomposition models play an increasingly important role in modern data science applications. One problem of particular interest is fitting a low-rank Canonical Polyadic (CP) tensor decomposition model when the tensor has sparse structure and the tensor elements are nonnegative count data. SparTen is a high-performance C++ library which computes a low-rank decomposition using different solvers: a first-order quasi-Newton or a second-order damped Newton method, along with the appropriate choice of runtime parameters. Since default parameters in SparTen are tuned to experimental results in prior published work on a single real-world dataset conducted using MATLAB implementations of these methods, it remains unclear if the parameter defaults in SparTen are appropriate for general tensor data. Furthermore, it is unknown how sensitive algorithm convergence is to changes in the input parameter values. This report addresses these unresolved issues with large-scale experimentation on three benchmark tensor data sets. Experiments were conducted on several different CPU architectures and replicated with many initial states to establish generalized profiles of algorithm convergence behavior.

1. Introduction. The Canonical Polyadic (CP) tensor decomposition model has garnered attention as a tool for extracting useful information from high dimensional data across a wide range of applications [10, 4, 9, 2, 8].

Recently, Hansen *et al.* developed two highly-parallelizable Newton-based methods for low-rank tensor factorizations on Poisson count data in [7], one a first-order quasi-Newton method (PQNR) and another a second-order damped Newton method (PDNR). These methods reformulate the CP Poisson tensor factorization optimization problem described in [3] into many independent subproblems that can be solved in parallel. They were first implemented in MATLAB Tensor Toolbox [1] as the function `cp_apr`, referring to this approach as computing a CP decomposition using Alternating Poisson Regression (i.e., CP-APR). These methods fit a reduced-rank CP model to count data, assuming a Poisson error distribution. PDNR and PQNR are implemented in SparTen,[1] a high-performance C++ library of CP-APR solvers for sparse tensors. SparTen improves on the MATLAB implementation to provide efficient execution for large, sparse tensor decompositions, exploiting the Kokkos hardware abstraction library [6] to harness parallelism on diverse HPC platforms, including x86-multicore, ARM, and GPU computer architectures.

SparTen contains many algorithmic parameters for controlling the optimization subroutines comprising PDNR and PQNR. To date, only anecdotal evidence exists for how best to tune the algorithms. Parameter defaults in SparTen were chosen according to previous results using the MATLAB implementations described by Hansen *et al.* [7]. However, their analysis was limited to a single real-world dataset, and thus may not be optimal for computing decompositions of more general tensor data. Furthermore, it is unknown how the initial guess to a solution affects convergence, since SparTen methods may converge slowly—or worse, stagnate—on real data if the initial state is far from a solution. And, lastly, the average impact of input parameters on algorithm convergence is unclear.

To address these unknowns, we present the results of numerical experiments to assess the sensitivity of software parameters on algorithm convergence for a range of values with benchmark tensor problems. Every experiment was replicated with 30 randomly chosen

*College of William and Mary, jmmyers01@email.wm.edu
†Sandia National Laboratories, dmdunla@sandia.gov
‡Sandia National Laboratories, knteran@sandia.gov
§Sandia National Laboratories, dshollm@sandia.gov

[1] SparTen is a portmanteau word derived from *Sparse* and *Tensor*. The SparTen code is available at `http://gitlab.com/tensors/sparten`.

initial guesses on three diverse computer architectures to aid statistical interpretation. With our results, we (1) provide new results that offer a realistic picture of algorithm convergence under reasonable resource constraints, (2) establish practical bounds on parameters such that, if set at or beyond these values, convergence is unlikely, and (3) identify areas of performance degradation and convergence toward qualitatively different results owing to parameter sensitivities.

We limited our study to multicore CPU architectures only, using OpenMP [5] to manage the parallel computations across threads/cores. Although SparTen, through Kokkos, can leverage other execution backends (e.g., NVIDIA's CUDA framework for GPU computation), we focus solely on diversity in CPU architectures in this work.

This paper is structured as follows. Section 2 summarizes basic tensor notation and details. Section 3 describes the hardware environment, test data, and experimental design of the sensitivity analysis. Section 4 provides detailed results of the sensitivity analyses. Section 5 offers concluding remarks and lays out future work.

2. Background. We briefly describe below the problem we are addressing in this report; for a detailed description of CP decomposition algorithms implemented in SparTen, refer to the descriptions in Hansen *et al.* [7].

An N-way data tensor $\mathcal{X}$ has dimension sizes $I_1 \times I_2 \times \cdots \times I_N$. We wish to fit a reduced-dimension tensor model, $\mathcal{M}$, to $\mathcal{X}$. The R-component Canonical Polyadic (CP) decomposition is given as follows:

$$\mathcal{X} \approx \mathcal{M} = [\![\boldsymbol{\lambda}; \mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}]\!] = \sum_{r=1}^{R} \lambda_r \mathbf{a}_r^{(1)} \circ \ldots \circ \mathbf{a}_r^{(N)}, \tag{2.1}$$

where $\boldsymbol{\lambda} = [\lambda_1, \ldots, \lambda_R]$ is a scaling vector, $\mathbf{a}_r^{(n)}$ represents the r-th column of the factor matrix $\mathbf{A}^{(n)}$ of size $I_n \times R$, and $\circ$ is the vector outer product. We refer to the operator $[\![\cdot]\!]$ as a Kruskal operator, and the tensor $\mathcal{M}$, with its specific multilinear model form, as a Kruskal tensor in (2.1). See [10] for more details regarding these definitions.

SparTen addresses the special case when the elements of $\mathcal{X}$ are nonnegative counts. Assuming the entries in $\mathcal{X}$ follow a Poisson distribution with multilinear parameters, the low-rank CP decomposition in (2.1) can be computed using the CP-APR methods, PDNR and PQNR, introduced by Hansen *et al.* [7].

3. Methods. In this section, we describe the hardware platforms, data, and SparTen algorithm parameters used in our experiments.

3.1. Hardware Platforms. We used diverse CPU architectures to perform our experiments, with hardware and compiler specifications detailed in Table 3.1. Intel 1–4 are production clusters with hundreds to thousands of nodes, whereas ARM and IBM clusters are advanced architecture research testbeds with tens of nodes each. We employed the maximum number of OpenMP threads available per node from each platform to maximize throughput and configured the maximum wall-clock limit as 12:00 hours for all experiments. Although all parallelism was solely across threads on a single node, we took advantage of the large number of nodes on each cluster to enable the large number of experiments. Thus, our analyses do not consider runtime. The latest GNU compiler available to each cluster, `gcc`, was used, with `-O3` optimization and Kokkos architecture-specific flags enabled.

3.2. Data. We experimented using sparse tensors of count data from the FROSTT collection [12]. Specifically, we chose the three datasets listed in Table 3.2 to account for size, dimensionality, and density (i.e., the ratio of nonzero entries to the total number of elements in the tensor):

Table 3.1: Hardware characteristics and software environment of the clusters in this paper. *Threads* and *RAM (GB)* are per node.

Arch	Processor	Threads	RAM (GB)	GCC
ARM	ThunderX2	256	255	7.2.0
IBM	Newell Power9	80	319	7.2.0
Intel 1	Sandy Bridge	16	64	8.2.1
Intel 2	Broadwell	72	128	8.2.1
Intel 3	Sandy Bridge	16	64	8.2.1
Intel 4	Sandy Bridge	32	64	8.2.1

1. Chicago Crime Community is a 4th-order tensor of crime reports in the city of Chicago spanning nearly 17 years. The four modes represent *day* × *hour* × *community* × *crime-type* and the values are counts.
2. LBNL-Network is a 5th-order tensor of anonymized network traffic at Lawrence Berkeley National Laboratory. The five modes represent *sender-IP* × *sender-port* × *destination-IP* × *destination-port* × *time* and the values are total packet length per timestep.
3. NELL-2 is 3rd-order benchmark tensor that gives a snapshot of the NELL: Never-Ending Language Learning relational database. The three modes represent *entity* × *relation* × *entity* relationships.

Throughout the discussion below, we refer to the data using the short names listed in the table.

Table 3.2: Sparse tensor datasets from the FROSTT collection.

FROSTT Name (short name)	Dimensions	Density
chicago-crime-comm (*chicago*)	$(6186, 24, 77, 32)$	1.5×10^{-02}
lbnl-network (*lbnl*)	$(1605, 4198, 1631, 4209, 868131)$	4.2×10^{-14}
nell-2 (*nell*)	$(12092, 9184, 28818)$	2.4×10^{-05}

3.3. Software Parameter Definitions & Experimental Ranges. PQNR and PDNR are composed of standard techniques in the numerical optimization literature. Specifically, for each tensor mode, the Newton optimization computes the gradient and Hessian matrix. Then, the inverse Hessian is approximated to compute a search direction and an Armijo backtracking line search is used to compute the Newton step. How the inverse Hessian is approximated differentiates PDNR and PQNR. PDNR shifts the eigenvalues by a damping factor μ to guarantee the Hessian matrix is semi-positive definite, and solves the resulting linear system exactly. PQNR approximates the inverse Hessian directly with a limited-memory BFGS (L-BFGS) approach, computed with a small number of update pairs. Since the algorithm parameters analyzed here are those presented in several equations and algorithms in [7], we defer to that paper for specific details.

A brief description of each software parameter is given below. We note that the stability parameters used to safeguard against numerical errors—e.g., offset tolerances to avoid

divide-by-zero floating point errors—do not appear in the corresponding Matlab Tensor Toolbox method `cp_apr`.

3.3.1. Parameter Descriptions.

- `max_outer_iterations`: Maximum number of outer iterations to perform (Algorithm 1, Steps 2-9 in [7]).
- `max_inner_iterations`: Maximum number of inner iterations to perform (K_{max} in Algorithms 3 and 4 in [7]).
- `max_backtrack_steps`: Maximum number of backtracking steps in line search (maximum allowable value of t used in Equation (17) in [7]).
- `min_step_length`: Tolerance for nonzero line search step length (smallest allowable value of β in Equation (17) in [7]).
- `step_reduction_factor`: Factor to reduce line search step size between iterations (β^{t+1}/β^t in Equation (17) in [7]).
- `suff_decrease_tolerance`: Tolerance to ensure the next iterate decreases the objective function (σ in Equation (17) in [7]).
- `mu_initial`: Initial value of damping parameter (μ_0 in Algorithm 3 in [7]).
- `damping_increase_factor`: Scalar value to increase damping parameter in next iterate (Equation (16) in [7]).
- `damping_decrease_factor`: Scalar value to decrease damping parameter in next iterate (Equation (16) in [7]).
- `damping_increase_tolerance`: Tolerance to increase the damping parameter in Equation (16) in [7]. If the search direction increases the objective function and the ratio of actual reduction and predicted reduction in objective function (ρ in Equation (15) in [7]) is less than `damping_increase_tolerance`, the damping parameter μ_k will be increased for the next iteration.
- `damping_decrease_tolerance`: Tolerance to decrease the damping parameter in Equation (16) in [7]. Conversely, if the search direction decreases the objective function and the ratio of actual reduction to predicted reduction (ρ in Equation (15) in [7]) is greater than `damping_decrease_tolerance`, the damping parameter μ_k will be decreased for the next iteration.
- `size_LBFGS`: Number of recent limited-BFGS (L-BFGS)-update pairs to use in estimating the current Hessian (M in Equation (18) in [7]).
- `eps_div_zero_grad`: Safeguard against divide-by-zero in gradient and Hessian calculations.
- `log_zero_safeguard`: Tolerance to avoid computing log(0) in objective function calculations.

The default value in SparTen of each parameter described above and the experimental ranges tested in these experiments are given in Table 3.3.

3.4. Experiments. An individual experiment is a job j on platform m solving a PDNR or PQNR row subproblem for dataset d with SparTen solver s, parameter p, parameter value v, and random initialization r; all remaining software parameters are fixed at the default values listed in Table 3.3. Certain experiments denoted with an asterisk* were run only on Intel hardware due to limited resources associated with the other architectures; this accounts for the larger number of experiments reported for these platforms. We conducted tests on these values to provide better resolution of the impact of the parameter where nearby values—i.e., on the bounds of the test range—contained uncertainty in the results. Furthermore, we split up the experiments across the Intel platforms by parameter, running the full set of experiments across all parameter values and all random initializations on a single platform. The superscripts denoted for each parameter in the table denote the

Table 3.3: SparTen software parameter descriptions and values used in our experiments.

Parameter	Default	Values Used in Experiments
`max_outer_iterations`[1]	10000	1, 2, 4, 8, 16, 32, 64, 128, 256, 512
`max_inner_iterations`[1]	20	20, 40, 80, 160
`max_backtrack_steps`[2]	10	1^*, 2, 4, 8, 10, 12^*, 16
`min_step_size`[2]	10^{-7}	10^{-1*}, 10^{-3}, 10^{-7}, 10^{-15*}
`step_reduction_factor`[2]	0.5	0.1, 0.3^*, 0.5, 0.7^*, 0.9
`suff_decrease_tol`[2]	10^{-4}	10^{-2}, 10^{-4}, 10^{-8*}, 10^{-12*}
`mu_initial`[3]	10^{-5}	10^{-2}, 10^{-5}, 10^{-8}
`damping_increase_factor`[3†]	3.5	1.5, 2.5^*, 3.5, 4.5^*, 5.5
`damping_decrease_factor`[3†]	2/7	0.1, 2/7, 0.3^*, 0.5, 0.7^*, 0.9
`damping_increase_tol`[3†]	0.25	0.1, 0.25, 0.495
`damping_decrease_tol`[3†]	0.75	0.505, 0.75, 0.9
`size_LBFGS`[3‡]	3	1, 2, 3, 4, 5, 10, 15, 20
`eps_div_zero_grad`[4]	10^{-10}	10^{-5}, 10^{-8*}, 10^{-10}, 10^{-12*}, 10^{-15}
`log_zero_safeguard`[4]	10^{-16}	10^{-4*}, 10^{-8}, 10^{-12*}, 10^{-16}, 10^{-24*}, 10^{-32}
`eps_active_set`[4]	$10^{-3\dagger}$	10^{-1}, 10^{-3}, 10^{-5*},10^{-8*}
	$10^{-8\ddagger}$	10^{-1}, 10^{-3}, 10^{-5*},10^{-8*}

[†]PDNR-specific; [‡]PQNR-specific; [1–4]Intel platform used for experiments; [*]values evaluated on Intel platform only

Intel platform number specified in Table 3.1. Since we report only the number of function evaluations and outer iterations in our results, we expect that running our experiments in this way has produced valid results.

In all experiments, we fit a 5-component CP decomposition using a tolerance of 10^{-4} (i.e., the value of τ in Equation (20) in [7], the violation of the Karush-Kuhn-Tucker (KKT) conditions, used as the stopping criterion for the methods we explore here). Computation of a CP decomposition using PDNR or PQNR in SparTen requires an initial guess of the model parameters—i.e., initial values for $\mathcal{M}$ in (2.1)—drawn from a uniform distribution in the range $[0, 1]$. As such, all experiments were replicated using 30 random initializations. All computations used double precision floating point arithmetic. We report results on the amount of computation required for convergence (i.e., the number of evaluations of the negative log likelihood objective function, $f(\mathcal{M})$, defined in Equation (4) of [7]) and the quality of the solution (i.e., the value of the negative log likelihood objective function). As each of our experiments consists of 30 replicates (i.e., 30 random initializations) across three CPU architectures, we report sample means and 95% confidence intervals (as defined in [11]) when presenting statistical trends in the results.

4. Results. In this section we analyze the results of the parameter sensitivity experiments and describe the statistical relationships between the convergence properties of the PDNR and PQNR methods and their input parameters.

In total, 21,960 unique experiments were planned, accounting for running PDNR and PQNR with random initializations across all parameter value ranges on the various hardware architectures described in Sections 3.1 and 3.3. An experiment *converged* if the final KKT violation is less than the value of $\tau = 10^{-4}$; an experiment reached *maximum iterations* if the number of outer iterations exceeded the maximum limit (i.e., `max_outer_iterations`) and did not converge; an experiment was *canceled* if it exceeded the wall-clock limit (i.e.,

SparTen neither converged to a solution nor reached maximum number of outer iterations within 12 hours); and an experiment was *missing* if it did not run due to a failure of the system to launch the experiment or other system issue. Of the planned experiments, we collected data from 16,139 experiments.

Table 4.1 presents the number of experiments planned (*plan.*) as defined above and the number of planned experiments where data was collected (i.e., planned minus missing). For those collected (*coll.*), the table shows the percentage of experiments that were canceled (*canc.*), converged (*conv.*), or exceeded the maximum iterations (*max. iter.*). We note that the most complete set of experiment results were obtained on the Intel platforms. Although there are many missing experiment results (*miss.*) for the IBM and ARM platforms, we attempt to identify patterns in the data we collected if there is strong evidence to support our claims. We note that a few parameters (`eps_active_set`, `min_step_size`, `suff_decrease_tol`, `damping_increase_tol`, `damping_decrease_tol`) showed no statistically significant differences across the range of input values used in the experiments. We conjecture that we did not find values where the parameters display sensitivities in the chosen tensor problems, thus it remains unclear if this behavior holds in general.

4.1. General Convergence Results on Real-World Data. As discussed in Section 1, applying PDNR and PQNR to real-world data has been explored previously in the literature only for a single problem. From Table 4.1, we observed that PQNR is canceled more than PDNR in the allotted time across datasets and CPU platforms. This confirms our intuition, since it is a classical result in iterative methods that damped Newton methods converge quadratically, in comparison to quasi-Newton methods, which converge superlinearly. Specifically, PQNR calls the objective function 2.7 times more than PDNR on average

Table 4.1: Experiments run on the different datasets and hardware platforms.

CPU	Solver	Data	Plan.	Collect.	Canc.	Conv.	Max. Iter.	Miss.
ARM	PDNR	*chicago*	1110	1110	4.8%	82.2%	13.0%	0.0%
		lbnl	1110	1110	10.5%	76.5%	13.0%	0.0%
		nell	1110	390	5.4%	39.2%	55.4%	64.9%
	PQNR	*chicago*	990	281	0.0%	55.5%	44.5	71.6%
		lbnl	990	237	0.0%	0.0%	100.0%	76.1%
		nell	990	390	23.3%	0.3%	76.4	60.6%
IBM	PDNR	*chicago*	1110	855	5.4%	77.8%	16.8%	23.0%
		lbnl	1110	692	11.3%	73.3%	15.4%	37.7%
		nell	1110	424	51.2%	12.0%	36.8%	61.8%
	PQNR	*chicago*	990	676	10.2%	76.3%	13.5%	31.7%
		lbnl	990	293	61.8%	0.0%	38.2%	70.4%
		nell	990	481	31.0%	6.6%	62.4%	51.4%
Intel	PDNR	*chicago*	1680	1673	5.0%	86.4%	8.6%	0.4%
		lbnl	1680	1663	11.0%	80.6%	8.4%	1.0%
		nell	1680	1643	44.7%	42.2%	13.1%	2.2%
	PQNR	*chicago*	1440	1434	12.1%	78.6%	9.3%	0.4%
		lbnl	1440	1363	78.0%	0.0%	22.0%	5.3%
		nell	1440	1424	68.8%	10.1%	21.1%	1.1%

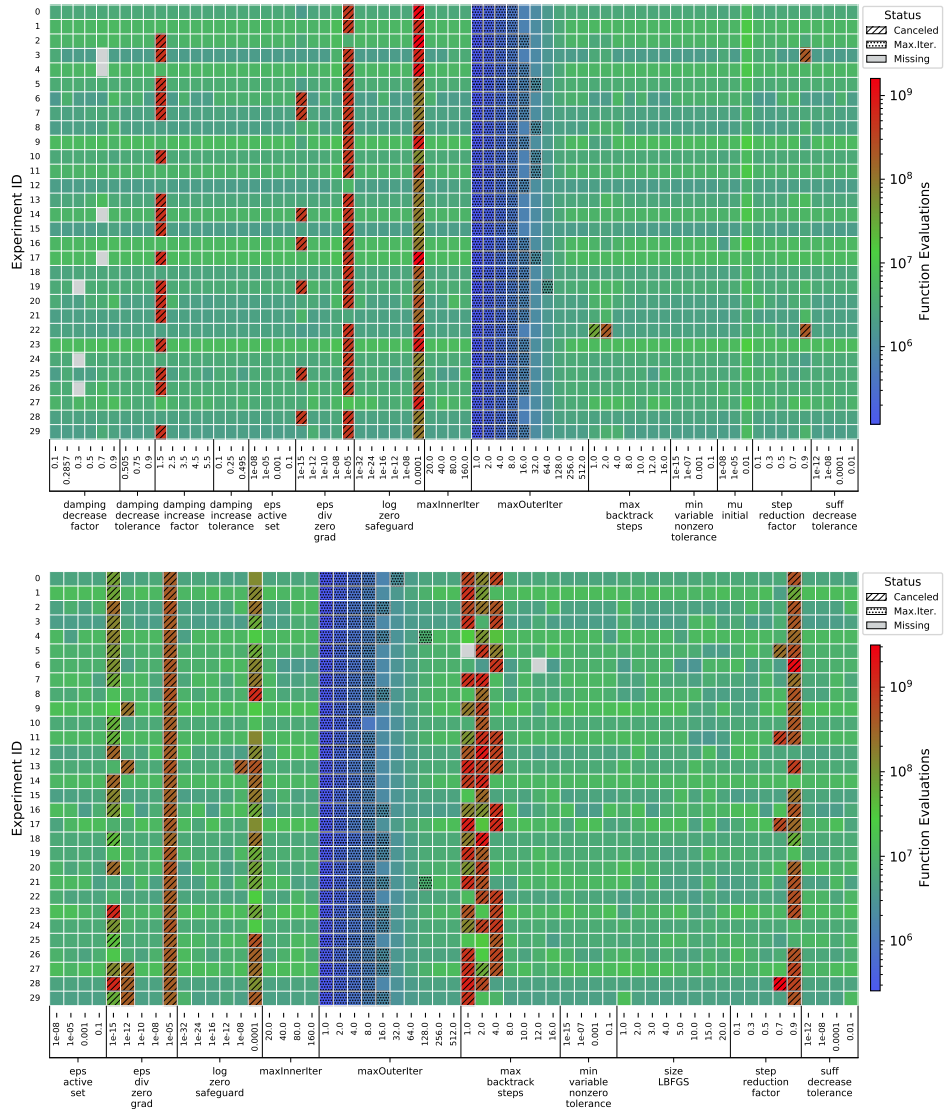


Fig. 4.1: Total function evaluations computed solving *chicago* on Intel architecture. Top: PDNR, Bottom: PQNR.

on the *chicago* data and fails to converge for any experiment on *lbnl* data across all hardware platforms. By contrast, PDNR converges in 86% of *lbnl* experiments across platforms when only 32 outer iterations are allowed.

4.2. Sensitivity of Convergence and Solution Behavior. There are certain ranges of parameter values that lead to good or bad convergence behaviors in general. This is illustrated in Figures 4.1–4.3, where parameter values and random initializations are depicted across the horizontal and vertical axes, respectively. These heatmaps display total objective function evaluations, where solid columns of a single shade indicate the same convergence

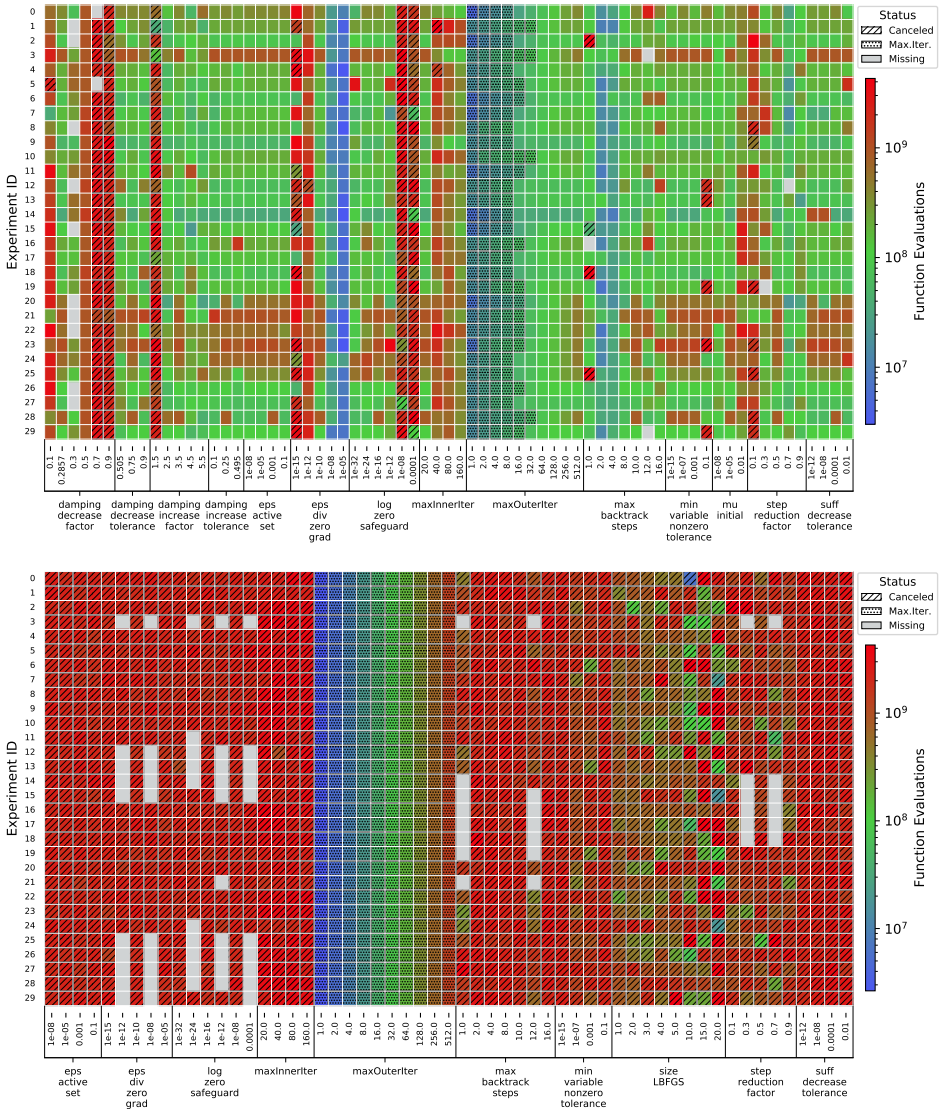


Fig. 4.2: Total function evaluations solving *lbnl* on Intel architecture. Top: PDNR, Bottom: PQNR.

behavior across all 30 random initializations. Green shades are consistent with *converged* experiments. Vertical bands not shaded green identify values that may impact algorithm performance, due either to iteration constraints (blue hues) or excessive computations corresponding to slow convergence or stagnation (red hues). Hatches denote non-converging exit status. Grey represents *missing* data, i.e., experiments that were planned but never conducted due to resource limitations—e.g., dequeued by the cluster administrator—or a system failure. Nearly solid column lines of the same shade indicate similar behavior, but also that there is some sensitivity of those parameter values to the initial starting point of the iterative methods.

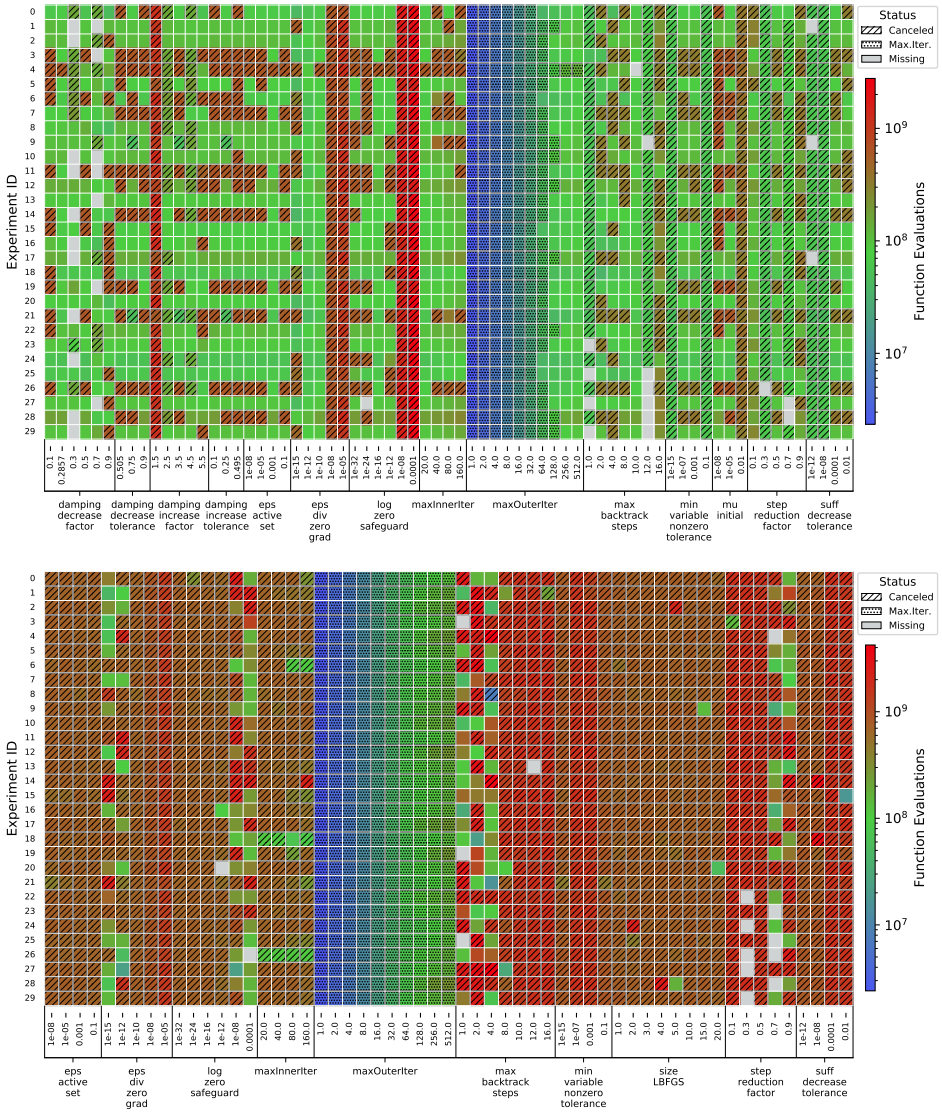


Fig. 4.3: Total function evaluations solving *nell* on Intel architecture. Top: PDNR, Bottom: PQNR.

In several cases, there is a tendency to time-out at one or both bounds of the test ranges for both solvers. The behavior of numerical stability parameters `eps_div_zero_grad` and `log_zero_safeguard` was consistent across combinations of solver, data, and CPU hardware. When `eps_div_zero_grad` is large, gradient directions that do not lead to objective function improvements may be scaled the same as gradient directions that do lead to such improvements. Furthermore, the corresponding eigenvalues of the Hessian matrix are amplified and Hessian information may be lost when determining the next iterate. For example, PDNR loses Hessian information as `eps_div_zero_grad` increases on *chicago* data; PDNR rarely converges and PQNR never converges when this parameter is relatively

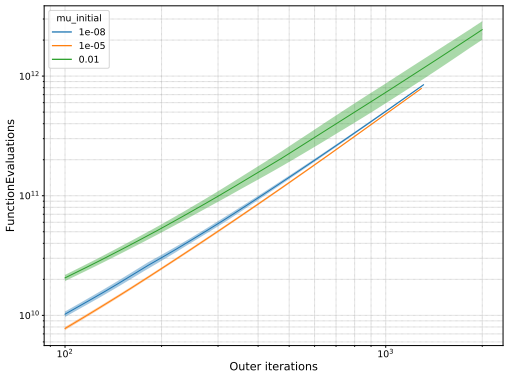


Fig. 4.4: PDNR damps out Hessian information and is more prone to time-outs when `mu_initial` is large (PDNR, *lbnl*).

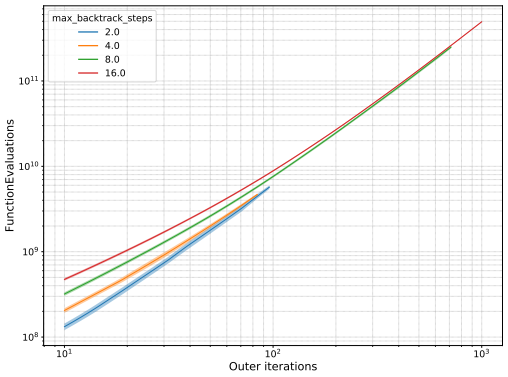


Fig. 4.5: The x-axis is truncated to emphasize the lower average cost when fewer `max_backtrack_steps` are allowed; the figure does not fully capture the high average cost when 16 maximum backtrack steps are allowed (PDNR, *lbnl*).

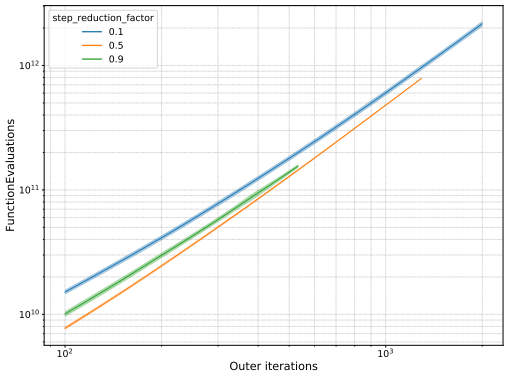


Fig. 4.6: The x-axis is truncated to demonstrate how high `step_reduction_factor` may accelerate convergence on large, sparse tensor data (PDNR, *lbnl*).

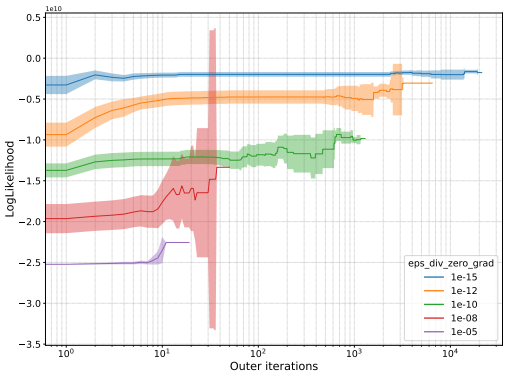


Fig. 4.7: Mean objective function values with 95% confidence interval, varying `eps_div_zero_grad` (PDNR, *lbnl*).

large—i.e. 10^{-5}. Moreover, both algorithms are sensitive to the parameter's lower bound, as small values may be insufficient to avoid an ill-conditioned Hessian matrix. In either case, additional iterations follow to correct errors incurred by `eps_div_zero_grad` values, large and small.

PDNR typically does not converge for large `log_zero_safeguard` values on large tensor problems. This parameter sets a nonzero offset in logarithm calculations to avoid explicitly computing log(0). High precision in logarithm computations tends to ensure the objective function is minimized accurately. When the value is too large, the calculated logarithm may be too small, and more backtracking steps are required to sufficiently decrease the objective function in the line search routine, making time-outs more likely. On the other

hand, the effect of the parameter on convergence is indistinguishable for values smaller than 10^{-8} across all experiments.

The effect when convergence behavior is similar for values set within sensitivity constraints is common among several algorithm parameters corresponding to the different numerical optimization subroutines that comprise PDNR and PQNR. Two examples are `damping_increase_factor` and `damping_decrease_factor`, which control updates to the PDNR Hessian matrix damping parameter μ. SparTen rarely converges when the former is set too low (1.5); the likely effect is that the updated damping factor is insufficient to guarantee a well-conditioned Hessian and too many unimportant directions are considered when computing the search direction. Above the 1.5 bound, the cost in objective function calls does not change significantly. The PQNR-specific parameter, `size_LBFGS`, behaves similarly; the only observable difference occurs when the update size is 1, using only the current iterate in the BFGS update.

Other parameters show meaningful differences in cost, defined in terms of the number of function evaluations required before convergence is achieved, when varied. Hansen *et al.* predict in [7] that when the damping parameter μ is set too large, a loss of Hessian information follows, which impacts convergence. For example, when `mu_initial` is large, the computational cost grows dramatically and time-outs become more likely, since the initial step length will at first be very small in every outer iteration and useful Hessian information is discarded in early stages of the inner loop solves. Convergence is most likely for a large, but not too-large, value, i.e., `mu_initial` $= 10^{-5}$. Cost grows 177.2% on *lbnl* and nearly doubles (+92.2%) on *chicago* as `mu_initial` grows from 10^{-5} to 10^{-2}. It is important to note in the former case that this cost is skewed by one experiment that converged after nearly 42,000 outer iterations, in comparison to 1,300 for the other parameter values on average, illustrated in Figure 4.4, where the x-axis is truncated to highlight the differences in total cost. Smaller values (i.e, 10^{-8}) seem to perform better for *chicago*, the smallest, densest problem and larger values (i.e., 10^{-5}) tend to perform better for large, sparse problems.

Allowing many backtracking steps during the line search may cause PDNR to waste effort; however, PQNR appears to perform better, in general, with more steps. PDNR is sensitive to the number of backtracking steps on *chicago*: average work performed is less when the maximum number of allowed steps is large and more work is performed when the number of steps is small. On *lbnl*—the sparsest tensor problem considered—PDNR performs better with fewer backtracking steps (see Figure 4.5). The average cost incurred by PQNR decreases as `max_backtrack_steps` increases.

The line search parameter `step_reduction_factor` is used to reduce the line search step size between iterations. On a large, sparse tensor problem, increasing this parameter may accelerate convergence. On the other hand, a small value makes convergence less certain. Figure 4.6 illustrates this behavior on the *lbnl* data: the average total cost decreased by 77% as `step_reduction_factor` increased from 0.1 to 0.5 (SparTen default) and decreased another 28% from 0.5 to 0.9. On *nell* data, PQNR *only* converged for large values (0.7, 0.9).

Parameter sensitivities affect not only convergence behavior, but may also produce qualitatively different results. Figure 4.7 illustrates the effect where large `eps_div_zero_grad`—and consequently, small step length—minimizes calls to the objective function *and* results in minimal objective function value. Most striking is that larger `eps_div_zero_grad` decreases the objective function more than an order of magnitude. This result was collected from 79 of 90 planned PDNR experiments on *lbnl*, and thus we consider this interesting effect worthy of further investigation.

5. Conclusions. Using results from more than 16,000 numerical experiments on several hardware platforms, we presented experimental results that expand our understanding of average PDNR and PQNR convergence on real-world tensor problems. We have shown that when using PQNR to compute large tensor decompositions convergence is less-likely under reasonable resource constraints. We have shown that some software parameters are sensitive to bounds on values. Further, we showed that varying several parameters can dramatically impact algorithm performance, and in some cases, may produce qualitatively different results.

Future work may address the issue of stagnation in Newton optimization methods for CP decompositions. We showed examples where the solver converged to a solution slowly but within the allotted time of 12 hours. For those experiments that timed out, it is unknown whether SparTen would eventually converge to a solution or stagnate without making progress. We anticipate that stagnation could be determined if the objective function values converge to a statistical steady state without satisfying the convergence criterion. Future development of SparTen may include dynamic updates to algorithm parameters based on local convergence information. Lastly, future experiments could explore coupled sensitivities among algorithm parameters, as this work was limited to single parameter, univariate analyses.

REFERENCES

[1] B. W. Bader, T. G. Kolda, et al., *Matlab tensor toolbox version 3.0-dev*. Available online, August 2017.

[2] J. Carroll and J. Chang, *Analysis of individual differences in multidimensional scaling via an n-way generalization of "eckart-young" decomposition*, Psychometrika, 35 (1970), pp. 283–319.

[3] E. C. Chi and T. G. Kolda, *On tensors, sparsity, and nonnegative factorizations*, SIAM Journal on Matrix Analysis and Applications, 33 (2012), pp. 1272–1299.

[4] A. Cichocki, D. Mandic, L. De Lathauwer, G. Zhou, Q. Zhao, C. Caiafa, and H. A. PHAN, *Tensor decompositions for signal processing applications: From two-way to multiway component analysis*, IEEE Signal Processing Magazine, 32 (2015), pp. 145–163.

[5] L. Dagum and R. Menon, *Openmp: an industry standard api for shared-memory programming*, Computational Science & Engineering, IEEE, 5 (1998), pp. 46–55.

[6] H. C. Edwards, C. R. Trott, and D. Sunderland, *Kokkos: Enabling manycore performance portability through polymorphic memory access patterns*, Journal of Parallel and Distributed Computing, 74 (2014), pp. 3202 – 3216. Domain-Specific Languages and High-Level Frameworks for High-Performance Computing.

[7] S. Hansen, T. Plantenga, and T. G. Kolda, *Newton-based optimization for Kullback-Leibler nonnegative tensor factorizations*, Optimization Methods and Software, 30 (2015), pp. 1002–1029.

[8] R. A. Harshman, *Foundations of the PARAFAC procedure: models and conditions for an "explanatory" multi-modal factor analysis*, UCLA Working Papers in Phonetics, 16 (1970), pp. 84–84.

[9] F. L. Hitchcock, *The expression of a tensor or a polyadic as a sum of products*, Journal of Mathematics and Physics, 6 (1927), pp. 164–189.

[10] T. Kolda and B. Bader, *Tensor decompositions and applications*, SIAM Review, 51 (2009), pp. 455–500.

[11] L. M. Leemis and S. K. Park, *Discrete-Event Simulation: A First Course*, Prentice-Hall, Inc., USA, 2005.

[12] S. Smith, J. W. Choi, J. Li, R. Vuduc, J. Park, X. Liu, and G. Karypis, *FROSTT: The formidable repository of open sparse tensors and tools*. Available online, 2017.

THE TEMPERED FRACTIONAL LAPLACIAN AS A SPECIAL CASE OF THE NONLOCAL LAPLACE OPERATOR

HAYLEY A. OLSON[†], MAMIKON GULIAN[‡], AND MARTA D'ELIA[§]

Abstract. Tempered fractional operators provide an improved predictive capability for modeling anomalous effects that cannot be captured by standard partial differential equations. These effects include subdiffusion and superdiffusion, which often occur in, e.g., geoscience and hydrology. Tempered operators can be used in such models of heavy-tailed behavior while circumventing consequences of standard fractional models, such as divergent moments. In the first part of this work, we investigate the relationship between tempered and truncated fractional operators and the unified nonlocal diffusion operator, building upon the recently developed unified nonlocal calculus. In the second part of this work, with the purpose of finding a computationally cheap alternative to tempered fractional operators, we investigate the relationship between the (computationally expensive) tempered fractional Laplacian and the (computationally cheaper) truncated fractional Laplacian. Our main result shows the equivalence between truncated and tempered fractional energies and represents the first step towards the approximation of expensive fractional models with cheaper, but equivalent, alternatives.

1. Introduction. Fractional models can capture anomalous effects that standard partial differential equations (PDEs) fail to describe. In particular, they can model superdiffusion and subdiffusion processes; i.e. processes for which the mean square displacement is proportional to time to a fractional power, instead of being linear with respect to time, as is the case for PDEs. These operators have been used for decades in subsurface diffusion and transport, where the anomalous behavior is caused by heterogeneities in materials or media [1, 5, 6, 12, 11], and have also found application in turbulence [7, 8, 10] and, more recently, in machine-learning algorithms [13].

Fractional operators, such as the fractional Laplacian, are integral operators acting on the whole space and, as such, feature infinite interactions between points or domains. This fact allows one to model long range forces and reduces the regularity requirements on the solution. However, despite their improved predictive capabilities, fractional models come with a high computational cost due to the infinite range of interactions and the singularities in their kernels. In this work we focus on the former matter and investigate an equivalent alternative to tempered fractional operators that is computationally cheaper. The alternative of choice is *truncated fractional operators*; i.e. fractional operators whose range of interaction is limited to a ball of finite radius.

In the first part of this work we investigate the relationship between tempered and truncated fractional operators and the unified nonlocal Laplacian operator, introduced in [3]. Specifically, we investigate the composition of tempered and truncated fractional divergence and gradient and compare it with the tempered and truncated fractional Laplacian operators. One of the contributions of this work is to show that, while for tempered case the composition yields a tempered fractional Laplacian, the same statement does not hold in the truncated case.

In the second part of the paper we focus on the equivalence between tempered and truncated fractional operators; our second main contribution is an equivalence result for the tempered and truncated energies. In particular, we show that for a given tempered parameter, the associated nonlocal energy norm is equivalent to the truncated energy norm for every truncation parameter.

This paper is organized as follows. In Section 2 we report relevant definitions and

[†]University of Nebraska-Lincoln, hayley.olson@huskers.unl.edu
[‡]Sandia National Laboratories, NM, mgulian@sandia.gov
[§]Sandia National Laboratories, CA, mdelia@sandia.gov

results that will be used throughout the paper. Section 3 introduces the tempered fractional Laplacian as a special case of a general nonlocal operator by using the nonlocal equivalence kernel from [3]. This is followed by Section 4 where we examine the same problem for the truncated fractional Laplacian. In Section 5, we investigate the relationship between the tempered and truncated fractional operators and prove the equivalence of the corresponding energies. Finally, in Section 6, we summarize our theoretical findings.

2. Notation and previous work. In this section we recall the definitions of unweighted and weighted nonlocal operators and the main result that will be useful throughout the paper.

We let $\Omega \in \mathbb{R}^n$ be an open bounded domain and define the corresponding *interaction domain* as

$$\begin{aligned}\Omega_I &= \{\mathbf{y} \in \mathbb{R}^n \backslash \Omega \text{ such that } \mathbf{x} \text{ interacts with } \mathbf{y} \text{ for some } \mathbf{x} \in \Omega\} \\ &= \{\mathbf{y} \in \mathbb{R}^n \backslash \Omega : |\mathbf{x}-\mathbf{y}| \leq \delta \text{ for some } \mathbf{x} \in \Omega\},\end{aligned}$$

where $\delta > 0$ is the so-called interaction radius or horizon. We point out that for fractional operators, including tempered fractional operators, $\delta = \infty$, so that $\Omega_I = \mathbb{R}^n \setminus \Omega$ (see the following section for a precise definition). Let $\mathbf{v} : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n$, $u : \mathbb{R}^n \to \mathbb{R}$, and let $\boldsymbol{\alpha} : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n$ be an antisymmetric function such that $supp(\boldsymbol{\alpha}) = B_\delta(\mathbf{x})$, for all $\mathbf{x} \in \mathbb{R}^n$, where $B_\delta(\mathbf{x})$ is the Euclidean ball of radius δ centered in $\mathbf{x}$. Then, for $\mathbf{x} \in \Omega$ the *nonlocal unweighted divergence and gradient* are defined as

$$\mathcal{D}\mathbf{v}(\mathbf{x}) := \int_{\Omega\cup\Omega_I} (\mathbf{v}(\mathbf{x},\mathbf{y}) + \mathbf{v}(\mathbf{y},\mathbf{x})) \cdot \boldsymbol{\alpha}(\mathbf{x},\mathbf{y}) d\mathbf{y} \tag{2.1}$$

$$\mathcal{G}u(\mathbf{x},\mathbf{y}) := (u(\mathbf{y}) - u(\mathbf{x}))\boldsymbol{\alpha}(\mathbf{x},\mathbf{y}). \tag{2.2}$$

The nonlocal unweighted Laplacian is obtained by composing the divergence and gradient operators, i.e. for $\mathbf{x} \in \Omega$ and $\gamma = \boldsymbol{\alpha} \cdot \boldsymbol{\alpha}$ the *nonlocal unweighted Laplacian* is defined as

$$\mathcal{L} = \mathcal{D}\mathcal{G}u(\mathbf{x}) = 2\int_{\Omega\cup\Omega_I} (u(\mathbf{y}) - u(\mathbf{x}))\gamma(\mathbf{x},\mathbf{y}) d\mathbf{y}.$$

Next, we let $\omega : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ be a symmetric two-point weight function; the *weighted nonlocal divergence and gradient* are defined as

$$\mathcal{D}_\omega \mathbf{v}(\mathbf{x}) := \mathcal{D}(\omega(\mathbf{x},\mathbf{y})\mathbf{v}(\mathbf{x})) = \int_{\Omega\cup\Omega_I} (\omega(\mathbf{x},\mathbf{y})\mathbf{v}(\mathbf{x}) + \omega(\mathbf{y},\mathbf{x})\mathbf{v}(\mathbf{y})) \cdot \boldsymbol{\alpha}(\mathbf{x},\mathbf{y}) d\mathbf{y} \tag{2.3}$$

$$\mathcal{G}_\omega u(\mathbf{x}) := \int_{\Omega\cup\Omega_I} \mathcal{G}u(\mathbf{x},\mathbf{y})\omega(\mathbf{x},\mathbf{y}) d\mathbf{y} = \int_{\Omega\cup\Omega_I} (u(\mathbf{y}) - u(\mathbf{x}))\boldsymbol{\alpha}(\mathbf{x},\mathbf{y})\omega(\mathbf{x},\mathbf{y}) d\mathbf{y}. \tag{2.4}$$

As above, for $\mathbf{x} \in \Omega$, by composing the weighted nonlocal divergence and gradient we obtain the *nonlocal weighted Laplacian*

$$\begin{aligned}\mathcal{L}_\omega u(\mathbf{x}) &= \mathcal{D}_\omega \mathcal{G}_\omega u(\mathbf{x}) \\ &= \int_{\Omega\cup\Omega_I} \left[\int_{\Omega\cup\Omega_I} (u(\mathbf{z}) - u(\mathbf{x}))\boldsymbol{\alpha}(\mathbf{x},\mathbf{z})\omega(\mathbf{x},\mathbf{z}) d\mathbf{z} \right. \\ &\qquad \left. + \int_{\Omega\cup\Omega_I} (u(\mathbf{z}) - u(\mathbf{y}))\boldsymbol{\alpha}(\mathbf{y},\mathbf{z})\omega(\mathbf{y},\mathbf{z}) d\mathbf{z} \right] \cdot \boldsymbol{\alpha}(\mathbf{x},\mathbf{y})\omega(\mathbf{x},\mathbf{y}) d\mathbf{y}.\end{aligned}$$

Theorem 4.1 in [3] shows that, given ω and $\boldsymbol{\alpha}$, it is possible to define the so-called *equivalence kernel*, γ_{eq}, for which the composition of weighted divergence and gradient equals the unweighted nonlocal Laplacian with kernel γ_{eq}. We report such result below.

THEOREM 2.1. *Let $\mathcal{D}_\omega$ and $\mathcal{G}_\omega$ be the operators associated with the symmetric weight function ω and the anti-symmetric function $\boldsymbol{\alpha}$. For the equivalence kernel γ_{eq} defined by*

$$\begin{aligned} 2\gamma_{eq}(\mathbf{x},\mathbf{y}) = \int_{\Omega\cup\Omega_I} & [\boldsymbol{\alpha}(\mathbf{x},\mathbf{y})\omega(\mathbf{x},\mathbf{y})\cdot\boldsymbol{\alpha}(\mathbf{x},\mathbf{z})\omega(\mathbf{x},\mathbf{z}) \\ & + \boldsymbol{\alpha}(\mathbf{z},\mathbf{y})\omega(\mathbf{z},\mathbf{y})\cdot\boldsymbol{\alpha}(\mathbf{x},\mathbf{y})\omega(\mathbf{x},\mathbf{y}) \\ & + \boldsymbol{\alpha}(\mathbf{z},\mathbf{y})\omega(\mathbf{z},\mathbf{y})\cdot\boldsymbol{\alpha}(\mathbf{x},\mathbf{z})\omega(\mathbf{x},\mathbf{z})]d\mathbf{z}, \end{aligned} \tag{2.5}$$

the weighted Laplacian $\mathcal{L}_\omega = \mathcal{D}_\omega\mathcal{G}_\omega$ and unweighted Laplacian operator $\mathcal{L}$ with kernel γ_{eq} are equivalent, i.e. $\mathcal{L} = \mathcal{L}_\omega$.

3. Tempered Fractional Laplacian as a Special Case of Nonlocal Operators. In this section we first show that for a specific choice of ω and $\boldsymbol{\alpha}$ the equivalence kernel is equivalent to the tempered fractional Laplacian kernel and then provide numerical illustrations that confirm the theoretical result. Throughout this section, we assume $u \in H^s(\mathbb{R}^n)$.

3.1. Consistency of tempered fractional Laplacian. The *tempered fractional Laplacian*, introduced in [9], is defined by

$$\mathcal{L}_{tem}u(\mathbf{x}) := \int_{\Omega\cup\Omega_I} (u(\mathbf{y}) - u(\mathbf{x}))\frac{e^{-\lambda|\mathbf{x}-\mathbf{y}|}}{|\mathbf{x}-\mathbf{y}|^{n+2s}}d\mathbf{y} \tag{3.1}$$

where $\lambda > 0$ and $0 < s < 1$ and where $\Omega \cup \Omega_I = \mathbb{R}^n$. Note that we do not consider a scaling constant (which usually appears in the literature for normalization purposes) as it is not relevant for the results reported in this paper. Also, while the integral above should be considered in a principal value sense, we do not explicitly write it in the definition of the operator, and implicitly assume it. Paper [3], shows that for

$$\begin{aligned} &\omega(\mathbf{x},\mathbf{y}) = |\mathbf{y}-\mathbf{x}|\phi(|\mathbf{y}-\mathbf{x}|) \;\text{ with }\; \phi(|\mathbf{y}-\mathbf{x}|) = \frac{e^{-\lambda|\mathbf{x}-\mathbf{y}|}}{|\mathbf{y}-\mathbf{x}|^{n+1+s}} \\ &\boldsymbol{\alpha}(\mathbf{x},\mathbf{y}) = \frac{\mathbf{y}-\mathbf{x}}{|\mathbf{y}-\mathbf{x}|} \end{aligned} \tag{3.2}$$

the equivalence kernel is given by

$$\gamma_{eq}(\mathbf{x},\mathbf{y}) = \frac{F(n,s,\lambda,|\mathbf{x}-\mathbf{y}|)}{|\mathbf{x}-\mathbf{y}|^{n+2s}} \tag{3.3}$$

for

$$F(n,s,\lambda,|\mathbf{x}-\mathbf{y}|) = \int_{\mathbb{R}^n} \frac{\mathbf{e}-\mathbf{z}}{|\mathbf{e}-\mathbf{z}|^{n+s+1}}\cdot\frac{\mathbf{z}}{|\mathbf{z}|^{n+s+1}}e^{-\lambda|\mathbf{x}-\mathbf{y}|(|\mathbf{e}-\mathbf{z}|+|\mathbf{z}|)}d\mathbf{z}. \tag{3.4}$$

In what follows, we make progress on the following conjecture, stated in [3] as Conjecture 4.1 for the special case of dimension $n = 1$.

CONJECTURE 3.1. *For the function F defined above, there exist positive constants $\underline{C}$ and $\overline{C}$ such that*

$$\underline{C}e^{-\lambda|\mathbf{x}-\mathbf{y}|} \le F(n,s,\lambda,|\mathbf{x}-\mathbf{y}|) \le \overline{C}e^{-\lambda|\mathbf{x}-\mathbf{y}|}. \tag{3.5}$$

Below, Lemma 3.2 establishes the lower bound for $n = 1$. Lemma 3.3 then proves a slightly weaker upper bound for the case $n = 1$, showing that the desired result holds for any $\lambda' > \lambda$. The latter lemma is shown using a linear approximation to the integrand in the conjecture. By using the same strategy and performing a more accurate approximation, it may be possible to prove the conjectured upper bound; we leave this to a future work.

LEMMA 3.2. *For the function F defined in (3.4), there exists a constant $\underline{C}$ such that $\underline{C}e^{-\lambda|\mathbf{x}-\mathbf{y}|} \leq F(n, s, \lambda, |\mathbf{x}-\mathbf{y}|)$. In particular,*

$$\underline{C} = \int_{-\infty}^{\infty} \frac{1-z}{|1-z|^{n+s+1}} \cdot \frac{z}{|z|^{n+s+1}} dz.$$

Proof. For $n = 1$, we analyze

$$F(n, s, \lambda, |x-y|) = \int_{-\infty}^{\infty} \frac{1-z}{|1-z|^{n+s+1}} \cdot \frac{z}{|z|^{n+s+1}} e^{-\lambda|x-y|(|1-z|+|z|)} dz.$$

Note that that the non-tempered integral

$$\int_{-\infty}^{\infty} \frac{1-z}{|1-z|^{n+s+1}} \cdot \frac{z}{|z|^{n+s+1}} dz$$

is a positive number [3]. The integrand of $F(n, s, \lambda, |x-y|)$ is nonnegative for $0 \leq z \leq 1$

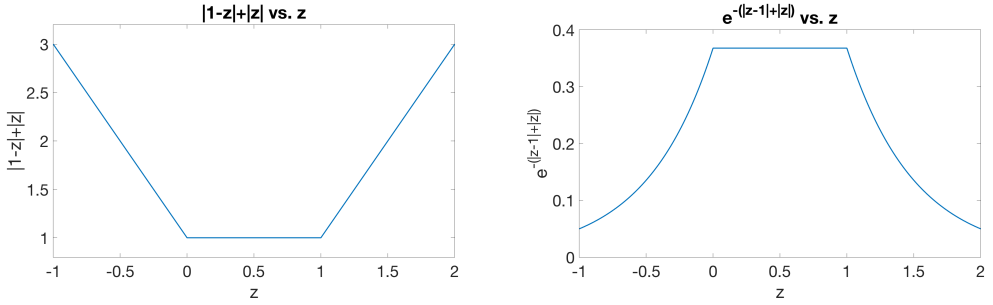


Fig. 3.1: **Left:** Plot of the function $f(z) = |z-1| + |z|$. **Right:** Plot of the exponential function $g(z) = e^{-\lambda|\mathbf{x}-\mathbf{y}|f(z)}$ for $\lambda|\mathbf{x}-\mathbf{y}| = 1$.

and negative elsewhere. We have

$$|1-z| + |z| = \begin{cases} 1-2z, & z < 0 \\ 1, & 0 \leq z \leq 1 \\ 2z-1, & z > 1. \end{cases}$$

Therefore,

$$e^{-\lambda|x-y|(|1-z|+|z|)} = \begin{cases} e^{-\lambda|x-y|(1-2z)} \leq e^{-\lambda|x-y|}, & z < 0 \\ e^{-\lambda|x-y|}, & 0 \leq z \leq 1 \\ e^{-\lambda|x-y|(2z-1)} \leq e^{-\lambda|x-y|}, & z > 1. \end{cases} \tag{3.6}$$

The integrand of $F(n, s, \lambda, |x-y|)$ for $0 \leq z \leq 1$, where the integrand is nonnegative, is

$$e^{-\lambda|x-y|} \int_{-\infty}^{\infty} \frac{1-z}{|1-z|^{n+s+1}} \cdot \frac{z}{|z|^{n+s+1}} dz.$$

Elsewhere, where the integrand is negative, the upper bounds on the exponential factor provide lower bounds for the integrand in exactly the same from. Thus, we obtain the lower bound

$$F(n, s, \lambda, |x-y|) \geq e^{-\lambda|x-y|} \int_{-\infty}^{\infty} \frac{1-z}{|1-z|^{n+s+1}} \cdot \frac{z}{|z|^{n+s+1}} dz = Ce^{-\lambda|x-y|}.$$

□

LEMMA 3.3. *For the function F defined in (3.4), for any $\lambda' > \lambda$ there is a constant $\overline{C}$ such that $F(n, s, \lambda, |\mathbf{x}-\mathbf{y}|) \leq \overline{C} e^{-\lambda'|\mathbf{x}-\mathbf{y}|}$.*

Proof. We observe that the factor

$$\frac{1-z}{|1-z|^{n+s+1}} \cdot \frac{z}{|z|^{n+s+1}}, \tag{3.7}$$

in the integrand of (3.4) is negative if $z < 0$ or $z > 1$ and nonnegative otherwise, while the remaining factor in the integrand is positive. Thus, lower bounds on the factor (3.6) of the integrand for $z < 0$ and $z > 1$ yield upper bounds on the integral (3.4). We claim that

$$e^{-\lambda|x-y|(|1-z|+|z|)} \geq \begin{cases} e^{-\lambda|x-y|}(2|x-y|z+1), & z < 0, \\ e^{-\lambda|x-y|}(2|x-y|(1-z)+1), & z > 1. \end{cases} \tag{3.8}$$

The second inequality follows from the first under the transformation $z \mapsto (1-z)$, which maps $\{z < 0\}$ to $\{z > 1\}$. To prove the first inequality, we denote

$$G(z) = e^{-\lambda|x-y|(|1-z|+|z|)}, \tag{3.9}$$

and note that for $z < 0$,

$$G(z) = e^{-\lambda|x-y|(1-2z)}. \tag{3.10}$$

Then

$$G'(z) = 2|x-y|e^{-\lambda|x-y|(1-2z)}, \tag{3.11}$$

$$G''(z) = 4|x-y|^2 e^{-\lambda|x-y|(1-2z)}. \tag{3.12}$$

Thus $G'(0) = 2|x-y|e^{-\lambda|x-y|}$, and $G''(z) > 0$ for all $z < 0$. Since $G(0) = e^{-\lambda|x-y|}$, it follows that for $z \leq 0$,

$$G(z) \geq G'(0)z + G(0) \tag{3.13}$$

$$\geq 2|x-y|e^{-\lambda|x-y|}z + e^{-\lambda|x-y|} \tag{3.14}$$

$$\geq e^{-\lambda|x-y|}(2|x-y|z+1). \tag{3.15}$$

This proves (3.8). Now define

$$a = \frac{1}{2|x-y|} \geq 0. \tag{3.16}$$

We have, for $z \le -a$,

$$e^{-\lambda|x-y|}(2|x-y|z+1) \le 0, \tag{3.17}$$

while for $z \ge 1+a$,

$$e^{-\lambda|x-y|}(2|x-y|(1-z)+1) \le 0. \tag{3.18}$$

Since $e^{-\lambda|x-y|(|1-z|+|z|)} > 0$ for all z, we can replace (3.8) by

$$e^{-\lambda|x-y|(|1-z|+|z|)} \ge \begin{cases} 0, & z \le -a, \\ e^{-\lambda|x-y|}(2|x-y|z+1), & -a < z < 0, \\ e^{-\lambda|x-y|}(2|x-y|(1-z)+1), & 1 < z < 1+a, \\ 0, & 1+a \le z. \end{cases} \tag{3.19}$$

From these inequalities and (3.4), we have

$$F(n,s,\lambda,|x-y|) \le \int_{-a}^{0} \frac{1-z}{|1-z|^{n+s+1}} \cdot \frac{z}{|z|^{n+s+1}} e^{-\lambda|x-y|}(2|x-y|z+1)dz \tag{3.20}$$

$$+ \int_{0}^{1} \frac{1-z}{|1-z|^{n+s+1}} \cdot \frac{z}{|z|^{n+s+1}} e^{-\lambda|x-y|}dz \tag{3.21}$$

$$+ \int_{1}^{1+a} \frac{1-z}{|1-z|^{n+s+1}} \cdot \frac{z}{|z|^{n+s+1}} e^{-\lambda|x-y|}(2|x-y|(1-z)+1)dz \tag{3.22}$$

$$= \int_{-a}^{0} \frac{1-z}{|1-z|^{n+s+1}} \cdot \frac{z}{|z|^{n+s+1}} e^{-\lambda|x-y|}2|x-y|zdz \tag{3.23}$$

$$+ \int_{-a}^{0} \frac{1-z}{|1-z|^{n+s+1}} \cdot \frac{z}{|z|^{n+s+1}} e^{-\lambda|x-y|}dz \tag{3.24}$$

$$+ \int_{0}^{1} \frac{1-z}{|1-z|^{n+s+1}} \cdot \frac{z}{|z|^{n+s+1}} e^{-\lambda|x-y|}dz \tag{3.25}$$

$$+ \int_{1}^{1+a} \frac{1-z}{|1-z|^{n+s+1}} \cdot \frac{z}{|z|^{n+s+1}} e^{-\lambda|x-y|}dz \tag{3.26}$$

$$+ \int_{1}^{1+a} \frac{1-z}{|1-z|^{n+s+1}} \cdot \frac{z}{|z|^{n+s+1}} e^{-\lambda|x-y|}2|x-y|(1-z)dz. \tag{3.27}$$

The second, third, and fourth terms above combine to give

$$e^{-\lambda|x-y|} \int_{-a}^{1+a} \frac{1-z}{|1-z|^{n+s+1}} \cdot \frac{z}{|z|^{n+s+1}} dz; \tag{3.28}$$

the integral in this expression is convergent due to the convergence of the improper integral

$$\int_{-\infty}^{\infty} \frac{1-z}{|1-z|^{n+s+1}} \cdot \frac{z}{|z|^{n+s+1}} dz \tag{3.29}$$

proven in [3]. Denote the value of the integral by C, so that the combination (3.28) can be

written as $Ce^{-\lambda|x-y|}$. The first term (3.23) can be evaluated as

$$\begin{aligned}\int_{-a}^{0} \frac{1-z}{|1-z|^{n+s+1}} \cdot \frac{z}{|z|^{n+s+1}} e^{-\lambda|x-y|} 2|x-y| z dz \\ = e^{-\lambda|x-y|} 2|x-y| \int_{-a}^{0} \frac{1-z}{(1-z)^{n+s+1}} \cdot \frac{|z|^2}{|z|^{n+s+1}} dz \\ = e^{-\lambda|x-y|} 2|x-y| \int_{-a}^{0} (1-z)^{-n-s} |z|^{-n+(1-s)} dz. \quad (3.30)\end{aligned}$$

This integral is improper due to the singularity at $z = 0$. Since we can assume that $s < 1$, we have $0 < 1 - s$, so that near $z \approx 0$ the integrand behaves as $z^{-n+\epsilon}$ for $\epsilon > 0$. Therefore, the integral converges and we can write the above as

$$C'|x-y|e^{-\lambda|x-y|} \quad (3.31)$$

for some constant C'. The fifth term (3.27) can be shown to satisfy the same upper bound using a similar calculation. Thus,

$$F(n, s, \lambda, |x-y|) \leq Ce^{-\lambda|x-y|} + C'|x-y|e^{-\lambda|x-y|}. \quad (3.32)$$

In turn, by a continuity and compactness argument, for any λ' there exists a constant $\overline{C}$ such that

$$Ce^{-\lambda|x-y|} + C'|x-y|e^{-\lambda|x-y|} \leq \overline{C}e^{-\lambda'|x-y|}. \quad (3.33)$$

This completes the proof. □

3.2. Numerical illustrations for the tempered fractional Laplacian. The expected behavior for F can be observed in the numerical illustrations presented in this section. Specifically, by displaying values of F in a semilog plot we observe slopes of value $-\lambda$, which indicates that F behaves like $e^{-\lambda|\mathbf{x}-\mathbf{y}|}$.

In Figures 3.2, 3.3, and 3.4, we report such plots for a fixed value for s.

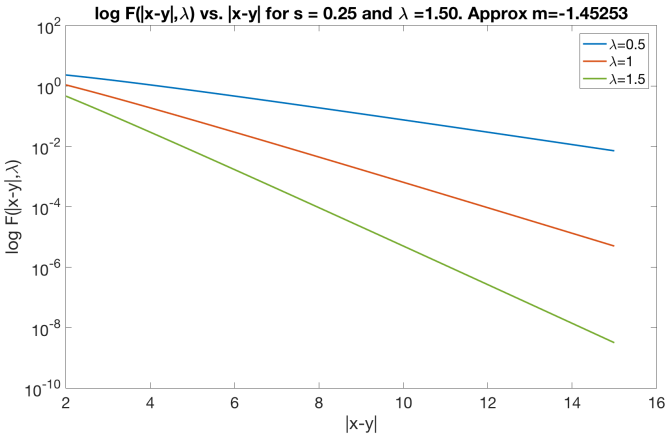


Fig. 3.2: Semilog plot of F vs. $|\mathbf{x} - \mathbf{y}|$ with s=0.25 fixed and varying $\lambda \in \{0.5, 1, 1.5\}$.

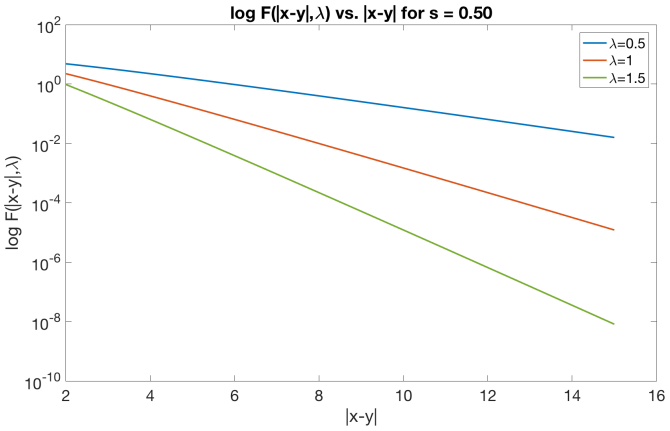


Fig. 3.3: Semilog plot of F vs. $|\mathbf{x}-\mathbf{y}|$ with s=0.5 fixed and varying $\lambda \in \{0.5, 1, 1.5\}$.

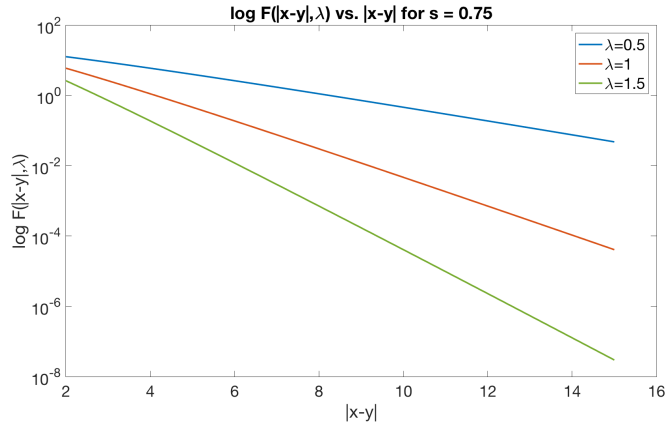


Fig. 3.4: Semilog plot of F vs. $|\mathbf{x}-\mathbf{y}|$ with s=0.75 fixed and varying $\lambda \in \{0.5, 1, 1.5\}$.

4. Truncated Fractional Laplacian as a Special Case of Nonlocal Operators. In this section we proceed as in the previous section and show that, as opposed to tempered operators, the composition of truncated divergence and gradient does not yield the truncated fractional Laplacian. We also provide numerical illustrations that confirm the theoretical result. Throughout this section, we assume $u \in H^s(\mathbb{R}^n)$.

4.1. Lack of equivalence kernel for the truncated fractional Laplacian. For $\mathbf{x} \in \Omega$, we define the *truncated fractional Laplacian* as

$$\mathcal{L}_{tr}u(\mathbf{x}) := \int_{\Omega \cup \Omega_I} (u(\mathbf{y}) - u(\mathbf{x})) \frac{\mathbb{1}\{|\mathbf{y}-\mathbf{x}| \le \delta\}}{|\mathbf{x}-\mathbf{y}|^{n+2s}} d\mathbf{y}. \tag{4.1}$$

Also in this case, we do not consider a scaling constant and we implicitly assume that the integral above is intended in a principal value sense.

In [3] the authors show that for a specific choice of $\boldsymbol{\alpha}$ and ω the composition of weighted divergence and gradient yields the fractional Laplacian operator. In this section we consider

the same functions and truncate the weight ω by multiplying it by the indicator function $\mathbb{1}\{|\mathbf{y}-\mathbf{x}|<\delta\}$. It is appealing to conjecture that by truncating the weight function over the ball of radius δ, the corresponding composition yields the truncated fractional Laplacian defined above. However, the following result shows that such a conjecture is not true.

THEOREM 4.1. *For $\boldsymbol{\alpha}$ and ω defined as*

$$\omega(\mathbf{x},\mathbf{y}) = |\mathbf{y}-\mathbf{x}|\phi(|\mathbf{y}-\mathbf{x}|) \;\; \textit{with} \;\; \phi(|\mathbf{y}-\mathbf{x}|) = \frac{\mathbb{1}\{|\mathbf{y}-\mathbf{x}|<\delta\}}{|\mathbf{y}-\mathbf{x}|^{n+1+s}} \tag{4.2}$$

$$\boldsymbol{\alpha}(\mathbf{x},\mathbf{y}) = \frac{\mathbf{y}-\mathbf{x}}{|\mathbf{y}-\mathbf{x}|}, \tag{4.3}$$

the equivalence kernel γ_{eq} has the form

$$\gamma_{eq} = \frac{1}{|\mathbf{x}-\mathbf{y}|^{n+2s}} F(|\mathbf{x}-\mathbf{y}|;\delta) \quad \textit{for}$$

$$F(|\mathbf{x}-\mathbf{y}|;\delta) = \int_{\mathbb{R}^n} \frac{\mathbf{e}-\mathbf{z}}{|\mathbf{e}-\mathbf{z}|^{n+s+1}} \frac{\mathbf{z}}{|\mathbf{z}|^{n+s+1}} \mathbb{1}\left\{|\mathbf{e}-\mathbf{z}| \le \frac{\delta}{|\mathbf{x}-\mathbf{y}|}\right\} \mathbb{1}\left\{|\mathbf{z}| \le \frac{\delta}{|\mathbf{x}-\mathbf{y}|}\right\} d\mathbf{z}.$$

Proof. With the choices above, we have

$$\begin{aligned}
\gamma_{eq}(\mathbf{x},\mathbf{y}) = \int_{\mathbb{R}^n} & [\boldsymbol{\alpha}(\mathbf{x},\mathbf{y})\omega(\mathbf{x},\mathbf{y}) \cdot \boldsymbol{\alpha}(\mathbf{x},\mathbf{z})\omega(\mathbf{x},\mathbf{z}) + \boldsymbol{\alpha}(\mathbf{z},\mathbf{y})\omega(\mathbf{z},\mathbf{y}) \cdot \boldsymbol{\alpha}(\mathbf{x},\mathbf{y})\omega(\mathbf{x},\mathbf{y}) \\
& + \boldsymbol{\alpha}(\mathbf{z},\mathbf{y})\omega(\mathbf{z},\mathbf{y}) \cdot \boldsymbol{\alpha}(\mathbf{x},\mathbf{z})\omega(\mathbf{x},\mathbf{z})] d\mathbf{z} \\
= \int_{\mathbb{R}^n} & \left[\frac{\mathbf{y}-\mathbf{x}}{|\mathbf{y}-\mathbf{x}|^{n+s+1}} \cdot \frac{\mathbf{z}-\mathbf{x}}{|\mathbf{z}-\mathbf{x}|^{n+s+1}} \mathbb{1}\{|\mathbf{y}-\mathbf{x}| \le \delta\}\mathbb{1}\{|\mathbf{z}-\mathbf{x}| \le \delta\} \right. \\
& + \frac{\mathbf{y}-\mathbf{z}}{|\mathbf{y}-\mathbf{z}|^{n+s+1}} \cdot \frac{\mathbf{y}-\mathbf{x}}{|\mathbf{y}-\mathbf{x}|^{n+s+1}} \mathbb{1}\{|\mathbf{y}-\mathbf{z}| \le \delta\}\mathbb{1}\{|\mathbf{y}-\mathbf{x}| \le \delta\} \\
& \left. + \frac{\mathbf{y}-\mathbf{z}}{|\mathbf{y}-\mathbf{z}|^{n+s+1}} \cdot \frac{\mathbf{z}-\mathbf{x}}{|\mathbf{z}-\mathbf{x}|^{n+s+1}} \mathbb{1}\{|\mathbf{y}-\mathbf{z}| \le \delta\}\mathbb{1}\{|\mathbf{z}-\mathbf{x}| \le \delta\} \right] d\mathbf{z}.
\end{aligned}$$

We rewrite the expression above as the sum of three terms, $\gamma_{eq}(\mathbf{x},\mathbf{y}) = I + II + III$ for

$$\begin{aligned}
I &= \mathbb{1}\{|\mathbf{y}-\mathbf{x}| \le \delta\} \frac{\mathbf{y}-\mathbf{x}}{|\mathbf{y}-\mathbf{x}|^{n+s+1}} \cdot \int_{\mathbb{R}^n} \frac{\mathbf{z}-\mathbf{x}}{|\mathbf{z}-\mathbf{x}|^{n+s+1}} \mathbb{1}\{|\mathbf{z}-\mathbf{x}| \le \delta\} d\mathbf{z}, \\
II &= \mathbb{1}\{|\mathbf{y}-\mathbf{x}| \le \delta\} \frac{\mathbf{y}-\mathbf{x}}{|\mathbf{y}-\mathbf{x}|^{n+s+1}} \cdot \int_{\mathbb{R}^n} \frac{\mathbf{y}-\mathbf{z}}{|\mathbf{y}-\mathbf{z}|^{n+s+1}} \mathbb{1}\{|\mathbf{y}-\mathbf{z}| \le \delta\} d\mathbf{z}, \\
III &= \int_{\mathbb{R}^n} \frac{\mathbf{y}-\mathbf{z}}{|\mathbf{y}-\mathbf{z}|^{n+s+1}} \cdot \frac{\mathbf{z}-\mathbf{x}}{|\mathbf{z}-\mathbf{x}|^{n+s+1}} \mathbb{1}\{|\mathbf{y}-\mathbf{z}| \le \delta\}\mathbb{1}\{|\mathbf{z}-\mathbf{x}| \le \delta\} d\mathbf{z}.
\end{aligned}$$

Note that $I = II = 0$ due to the rotational symmetry of the integrand. Thus, the truncated kernel is given by

$$\gamma_{eq} = \int_{\mathbb{R}^d} \frac{\mathbf{y}-\mathbf{z}}{|\mathbf{y}-\mathbf{z}|^{n+s+1}} \cdot \frac{\mathbf{z}-\mathbf{x}}{|\mathbf{z}-\mathbf{x}|^{n+s+1}} \mathbb{1}\{|\mathbf{y}-\mathbf{z}| \le \delta\}\mathbb{1}\{|\mathbf{z}-\mathbf{x}| \le \delta\} d\mathbf{z}.$$

We can apply the change of variables $\mathbf{z} \mapsto \mathbf{z}+\mathbf{x}$ to obtain

$$\gamma_{eq} = \int_{\mathbb{R}^d} \frac{(\mathbf{y}-\mathbf{x})-\mathbf{z}}{|(\mathbf{y}-\mathbf{x})-\mathbf{z}|^{n+s+1}} \cdot \frac{\mathbf{z}}{|\mathbf{z}|^{n+s+1}} \mathbb{1}\{|(\mathbf{y}-\mathbf{x})-\mathbf{z}| \le \delta\}\mathbb{1}\{|\mathbf{z}| \le \delta\} d\mathbf{z}.$$

This demonstrates that K only depends on $(\mathbf{y}-\mathbf{x})$. Next, we apply a rotation $\mathcal{R}$ and compute

$$\gamma_{eq}\left(\mathcal{R}(\mathbf{x}-\mathbf{y})\right) = \int_{\mathbb{R}^n} \frac{\mathcal{R}(\mathbf{y}-\mathbf{x})-\mathbf{z}}{|\mathcal{R}(\mathbf{y}-\mathbf{x})-\mathbf{z}|^{n+s+1}} \cdot \frac{\mathbf{z}}{|\mathbf{z}|^{n+s+1}} \mathbb{1}\{|\mathcal{R}(\mathbf{y}-\mathbf{x})-\mathbf{z}| \le \delta\}\mathbb{1}\{|\mathbf{z}| \le \delta\} d\mathbf{z}.$$

Let $\mathbf{z} = \mathcal{R}\mathbf{z}_{\text{new}}$. Then $d\mathbf{z} = d\mathbf{z}_{\text{new}}$, and

$$\begin{aligned}
\gamma_{eq}\left(\mathcal{R}(\mathbf{x}-\mathbf{y})\right) &= \int_{\mathbb{R}^n} \frac{\mathcal{R}(\mathbf{y}-\mathbf{x})-\mathcal{R}\mathbf{z}_{\text{new}}}{|\mathcal{R}(\mathbf{y}-\mathbf{x})-\mathcal{R}\mathbf{z}_{\text{new}}|^{n+s+1}} \cdot \frac{\mathcal{R}\mathbf{z}_{\text{new}}}{|\mathcal{R}\mathbf{z}_{\text{new}}|^{n+s+1}} \\
&\qquad \mathbb{1}\{|\mathcal{R}(\mathbf{y}-\mathbf{x})-\mathcal{R}\mathbf{z}| \le \delta\}\mathbb{1}\{|\mathcal{R}\mathbf{z}| \le \delta\} d\mathbf{z}_{\text{new}} \\
&= \int_{\mathbb{R}^n} \frac{\mathcal{R}(\mathbf{y}-\mathbf{x})-\mathcal{R}\mathbf{z}}{|\mathcal{R}(\mathbf{y}-\mathbf{x})-\mathcal{R}\mathbf{z}|^{n+s+1}} \cdot \frac{\mathcal{R}\mathbf{z}}{|\mathcal{R}\mathbf{z}|^{n+s+1}} \\
&\qquad \mathbb{1}\{|\mathcal{R}(\mathbf{y}-\mathbf{x})-\mathcal{R}\mathbf{z}| \le \delta\}\mathbb{1}\{|\mathcal{R}\mathbf{z}| \le \delta\} d\mathbf{z} \\
&= \int_{\mathbb{R}^n} \frac{\mathcal{R}\left((\mathbf{y}-\mathbf{x})-\mathbf{z}\right)}{|\mathcal{R}\left((\mathbf{y}-\mathbf{x})-\mathbf{z}\right)|^{n+s+1}} \cdot \frac{\mathcal{R}\mathbf{z}}{|\mathcal{R}\mathbf{z}|^{n+s+1}} \\
&\qquad \mathbb{1}\{|\mathcal{R}(\mathbf{y}-\mathbf{x})-\mathcal{R}\mathbf{z}| \le \delta\}\mathbb{1}\{|\mathcal{R}\mathbf{z}| \le \delta\} d\mathbf{z} \\
&= \int_{\mathbb{R}^n} \frac{1}{|\mathcal{R}\left((\mathbf{y}-\mathbf{x})-\mathbf{z}\right)|^{n+s+1}} \frac{1}{|\mathcal{R}\mathbf{z}|^{n+s+1}} \left[\mathcal{R}\left((\mathbf{y}-\mathbf{x})-\mathbf{z}\right)\cdot \mathcal{R}\mathbf{z}\right] \\
&\qquad \mathbb{1}\{|\mathcal{R}(\mathbf{y}-\mathbf{x}-\mathbf{z})| \le \delta\}\mathbb{1}\{|\mathcal{R}\mathbf{z}| \le \delta\} d\mathbf{z} \\
&= \int_{\mathbb{R}^n} \frac{1}{|\left((\mathbf{y}-\mathbf{x})-\mathbf{z}\right)|^{n+s+1}} \frac{1}{|\mathbf{z}|^{n+s+1}} \left[\left((\mathbf{y}-\mathbf{x})-\mathbf{z}\right)\cdot \mathbf{z}\right] \\
&\qquad \mathbb{1}\{|\mathbf{y}-\mathbf{x}-\mathbf{z}| \le \delta\}\mathbb{1}\{|\mathbf{z}| \le \delta\} d\mathbf{z} \\
&= \int_{\mathbb{R}^n} \frac{(\mathbf{y}-\mathbf{x})-\mathbf{z}}{|(\mathbf{y}-\mathbf{x})-\mathbf{z}|^{n+s+1}} \cdot \frac{\mathbf{z}}{|\mathbf{z}|^{n+s+1}} \\
&\qquad \mathbb{1}\{|\mathbf{y}-\mathbf{x}-\mathbf{z}| \le \delta\}\mathbb{1}\{|\mathbf{z}| \le \delta\} d\mathbf{z} \\
&= K(\mathbf{x}-\mathbf{y}).
\end{aligned}$$

This demonstrates that the truncated kernel is a rotationally invariant function of $\mathbf{y}-\mathbf{x}$; that is, it is a function just of the scalar norm of $\mathbf{y}-\mathbf{x}$. Next, we study the scaling by computing

$$\gamma_{eq}(c|\mathbf{x}-\mathbf{y}|;\delta) = \int_{\mathbb{R}^n} \frac{c(\mathbf{y}-\mathbf{x})-\mathbf{z}}{|c(\mathbf{y}-\mathbf{x})-\mathbf{z}|^{n+s+1}} \cdot \frac{\mathbf{z}}{|\mathbf{z}|^{n+s+1}} \mathbb{1}\{|c(\mathbf{y}-\mathbf{x})-\mathbf{z}| \le \delta\}\mathbb{1}\{|\mathbf{z}| \le \delta\} d\mathbf{z}.$$

for $c > 0$. Let $\mathbf{z} = c\mathbf{z}_{\text{new}}$. Then $d\mathbf{z} = c^n d\mathbf{z}_{\text{new}}$, and

$$\begin{aligned}
\gamma_{eq}(c(\mathbf{x}-\mathbf{y});\delta) &= \int_{\mathbb{R}^n} \frac{c(\mathbf{y}-\mathbf{x}) - c\mathbf{z}_{\text{new}}}{|c(\mathbf{y}-\mathbf{x}) - c\mathbf{z}_{\text{new}}|^{n+s+1}} \cdot \frac{c\mathbf{z}_{\text{new}}}{|c\mathbf{z}_{\text{new}}|^{n+s+1}} \\
&\qquad \mathbb{1}\{|c(\mathbf{y}-\mathbf{x}) - c\mathbf{z}_{\text{new}}| \le \delta\}\mathbb{1}\{|c\mathbf{z}_{\text{new}}| \le \delta\} c^n d\mathbf{z}_{\text{new}} \\
&= \int_{\mathbb{R}^n} \frac{c(\mathbf{y}-\mathbf{x}) - c\mathbf{z}}{|c(\mathbf{y}-\mathbf{x}) - c\mathbf{z}|^{n+s+1}} \cdot \frac{c\mathbf{z}}{|c\mathbf{z}|^{n+s+1}} \\
&\qquad \mathbb{1}\{c|(\mathbf{y}-\mathbf{x}) - \mathbf{z}| \le \delta\}\mathbb{1}\{c|\mathbf{z}| \le \delta\} c^n d\mathbf{z} \\
&= \frac{c}{c^{n+s+1}} \frac{c}{c^{n+s+1}} c^n \int_{\mathbb{R}^n} \frac{(\mathbf{y}-\mathbf{x}) - \mathbf{z}}{|(\mathbf{y}-\mathbf{x}) - \mathbf{z}|^{n+s+1}} \cdot \frac{\mathbf{z}}{|\mathbf{z}|^{n+s+1}} \\
&\qquad \mathbb{1}\{c|(\mathbf{y}-\mathbf{x}) - \mathbf{z}| \le \delta\}\mathbb{1}\{c|\mathbf{z}| \le \delta\} d\mathbf{z} \\
&= \frac{1}{c^{n+2s}} \int_{\mathbb{R}^n} \frac{(\mathbf{y}-\mathbf{x}) - \mathbf{z}}{|(\mathbf{y}-\mathbf{x}) - \mathbf{z}|^{n+s+1}} \cdot \frac{\mathbf{z}}{|\mathbf{z}|^{n+s+1}} \\
&\qquad \mathbb{1}\left\{|(\mathbf{y}-\mathbf{x}) - \mathbf{z}| \le \frac{\delta}{c}\right\} \mathbb{1}\left\{|\mathbf{z}| \le \frac{\delta}{c}\right\} d\mathbf{z}.
\end{aligned}$$

Then, we write

$$\begin{aligned}
\gamma_{eq}(|\mathbf{x}-\mathbf{y}|;\delta) &= \gamma_{eq}\left(|\mathbf{x}-\mathbf{y}|\frac{\mathbf{x}-\mathbf{y}}{|\mathbf{x}-\mathbf{y}|};\delta\right) \\
&= K\left(|\mathbf{x}-\mathbf{y}|\mathbf{e};\mathbf{ffi}\right) \\
&= \frac{1}{|\mathbf{x}-\mathbf{y}|^{n+2s}} \int_{\mathbb{R}^n} \frac{\mathbf{e}-\mathbf{z}}{|\mathbf{e}-\mathbf{z}|^{\mathbf{n+s+1}}} \\
&\qquad \cdot \frac{\mathbf{z}}{|\mathbf{z}|^{n+s+1}} \mathbb{1}\left\{|\mathbf{e}-\mathbf{z}| \le \frac{\mathbf{ffi}}{|\mathbf{x}-\mathbf{y}|}\right\} \mathbb{1}\left\{|\mathbf{z}| \le \frac{\delta}{|\mathbf{x}-\mathbf{y}|}\right\} d\mathbf{z} \\
&= \frac{1}{|\mathbf{x}-\mathbf{y}|^{n+2s}} F(|\mathbf{x}-\mathbf{y}|;\delta).
\end{aligned}$$

□

By analyzing the form of F, this theorem shows that the equivalence kernel for the truncated fractional weight exhibits unbounded behavior when $|\mathbf{x}-\mathbf{y}| = \delta$ in addition to when $|\mathbf{x}-\mathbf{y}| = 0$, which is not observed in the kernel of the truncated Laplacian. In the next section we illustrate this result using numerical computations of the equivalence kernel in the above theorem.

4.2. Numerical results for the truncated fractional Laplacian. In Figures 4.1 and 4.2 we report the functions F and K above; these plots confirm the singular behavior at $|\mathbf{x}-\mathbf{y}| = \delta$ and confirm that the composition of truncated fractional divergence and gradient is not consistent with the truncated fractional Laplacian.

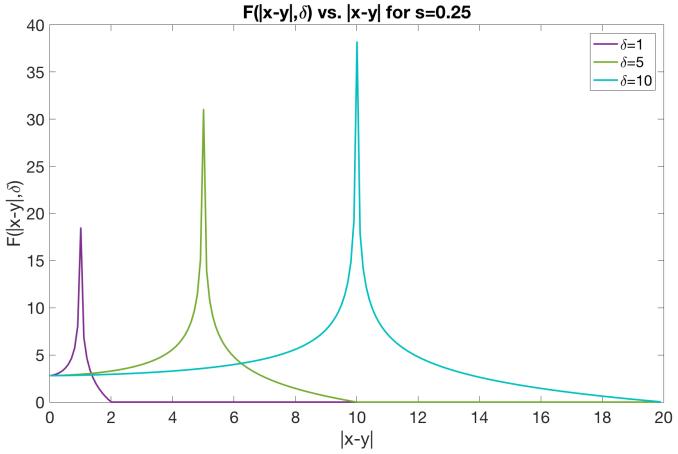


Fig. 4.1: Plot of F vs. $|\mathbf{x}-\mathbf{y}|$ with s=0.25 fixed and varying $\delta \in \{1, 5, 10\}$. Note the singularities at $|\mathbf{x}-\mathbf{y}| = \delta$.

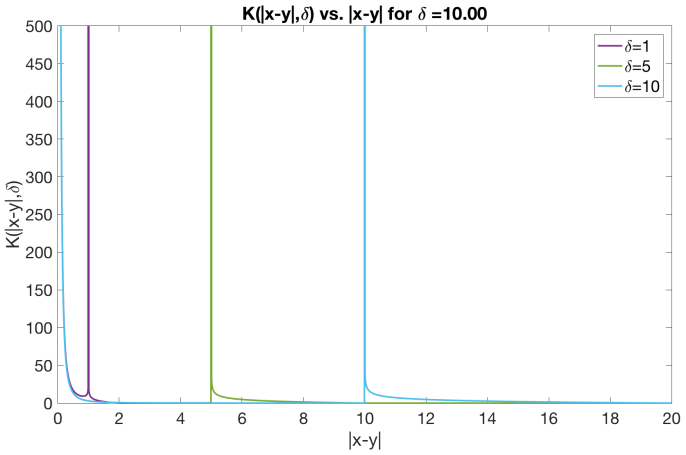


Fig. 4.2: Plot of K vs. $|\mathbf{x}-\mathbf{y}|$ with s=0.25 fixed and varying $\delta \in \{1, 5, 10\}$. While all three equivalence kernels exhibit singularities at $|\mathbf{x}-\mathbf{y}| = 0$ as expected, they are also singular at the respective values of $|\mathbf{x}-\mathbf{y}| = \delta$.

5. Equivalence of Tempered and Truncated Fractional Laplacians. In this section we investigate the relationship between the truncated and tempered fractional Laplacians. Our main goal is to find a viable, but equivalent, alternative to tempered fractional operators, that, due to their infinite interaction range, are extremely computationally expensive. Specifically, we compare the tempered and truncated fractional energy norms and show that given a tempered parameter λ, the associated energy is equivalent to a truncated fractional energy for any truncation parameter δ.

Throughout this section we consider functions $u \in H^s(\mathbb{R}^n)$ such that $u = 0$ in $\mathbb{R}^n \setminus \Omega$ and we refer to this functional space as $H^s_\Omega(\mathbb{R}^n)$. This assumption, though not necessary,

simplifies the analysis. For a kernel γ_i, the *nonlocal energy norm* is defined as

$$E_i(u;\mu) = \iint_{(\Omega\cup\Omega_I)^2} (u(\mathbf{x}) - u(\mathbf{y}))^2 \gamma_i(\mathbf{x},\mathbf{y},\mu) d\mathbf{y} d\mathbf{x} \tag{5.1}$$

where μ is a parameter that determines the kernel. We recall that for the tempered and truncated fractional Laplacian operators the kernel γ_i is defined as

$$\gamma_{tem}(\mathbf{x},\mathbf{y},\lambda) = \frac{e^{-\lambda|\mathbf{x}-\mathbf{y}|}}{|\mathbf{x}-\mathbf{y}|^{n+2s}} \quad \text{and} \quad \gamma_{tr}(\mathbf{x},\mathbf{y},\delta) = \frac{\mathbb{1}\{|\mathbf{x}-\mathbf{y}|<\delta\}}{|\mathbf{x}-\mathbf{y}|^{n+2s}}, \tag{5.2}$$

respectively. We refer to the corresponding energy norms $E_{tem}(u;\lambda)$ and $E_{tr}(u;\delta)$ as the tempered and truncated energies. Furthermore, by definition of the interaction domain Ω_I, for the tempered fractional Laplacian $\Omega_I = \mathbb{R}^n \setminus \Omega$, whereas for the truncated fractional Laplacian $\Omega_I = \{\mathbf{y} \in \mathbb{R}^n \setminus \Omega : \ |\mathbf{x}-\mathbf{y}| \le \delta, \text{ for some } \mathbf{x} \in \Omega\}$.

In what follows, we show that there exist positive constants $\underline{A}$ and $\overline{A}$ such that, given $\lambda > 0$,

$$\underline{A} E_{tr}(u;\delta) \le E_{tem}(u;\lambda) \le \overline{A} E_{tr}(u;\delta), \quad \forall\, u \in H^s_\Omega(\mathbb{R}^n),\ \delta < \infty. \tag{5.3}$$

The following theorem provides an estimate for the left-hand side of the inequality above.

THEOREM 5.1. *For the nonlocal truncated and tempered energies, the left-hand side of (5.3) holds with* $\underline{A} = e^{-\lambda\delta}$.

Proof. Due to the positivity of the integrand,

$$\begin{aligned} E_{tr}(u;\delta) &= \int_{\Omega\cup\Omega_I}\int_{\Omega\cup\Omega_I} (u(\mathbf{x}) - u(\mathbf{y}))^2 \gamma_{tr}(\mathbf{x},\mathbf{y},\delta) d\mathbf{y} d\mathbf{x} \\ &= \int_{\Omega\cup\Omega_I}\int_{\Omega\cup\Omega_I} (u(\mathbf{x}) - u(\mathbf{y}))^2 \frac{\chi\{|\mathbf{x}-\mathbf{y}|<\delta\}}{|\mathbf{x}-\mathbf{y}|^{n+2s}} d\mathbf{y} d\mathbf{x} \\ &\le \int_{\mathbb{R}^n}\int_{\mathbb{R}^n} (u(\mathbf{x}) - u(\mathbf{y}))^2 \frac{\chi\{|\mathbf{x}-\mathbf{y}|<\delta\}}{|\mathbf{x}-\mathbf{y}|^{n+2s}} d\mathbf{y} d\mathbf{x}. \end{aligned}$$

Note that for all $\mathbf{x},\mathbf{y} \in \mathbb{R}^n$,

$$e^{\lambda\delta} e^{-\lambda|\mathbf{x}-\mathbf{y}|} \ge \chi\{|\mathbf{x}-\mathbf{y}|<\delta\}.$$

Thus,

$$\begin{aligned} E_{tr}(u;\delta) &\le \int_{\mathbb{R}^n}\int_{\mathbb{R}^n} (u(\mathbf{x}) - u(\mathbf{y}))^2 \frac{\chi\{|\mathbf{x}-\mathbf{y}|<\delta\}}{|\mathbf{x}-\mathbf{y}|^{n+2s}} d\mathbf{y} d\mathbf{x} \\ &\le \int_{\mathbb{R}^n}\int_{\mathbb{R}^n} (u(\mathbf{x}) - u(\mathbf{y}))^2 \frac{e^{\lambda\delta} e^{-\lambda|\mathbf{x}-\mathbf{y}|}}{|\mathbf{x}-\mathbf{y}|^{n+2s}} d\mathbf{y} d\mathbf{x} \\ &= e^{\lambda\delta} \int_{\mathbb{R}^n}\int_{\mathbb{R}^n} (u(\mathbf{x}) - u(\mathbf{y}))^2 \frac{e^{-\lambda|\mathbf{x}-\mathbf{y}|}}{|\mathbf{x}-\mathbf{y}|^{n+2s}} d\mathbf{y} d\mathbf{x} \\ &= e^{\lambda\delta} E_{tem}(u;\lambda). \end{aligned}$$

□

In the remainder of this section we provide several results yielding the estimate on the right-hand side of inequality (5.3).

The following lemma shows that the integration domain of the truncated energy can be extended to $(\mathbb{R}^n)^2$.

LEMMA 5.2. *For $u \in H^s_\Omega(\mathbb{R}^n)$,*

$$E_{tr}(u;\delta) = \iint_{(\mathbb{R}^n)^2} (u(\mathbf{x}) - u(\mathbf{y}))^2 |\mathbf{x}-\mathbf{y}|^{-n-2s} \mathbb{1}\{|\mathbf{x}-\mathbf{y}| \le \delta\} d\mathbf{x} d\mathbf{y}.$$

Proof. We let $G = G(\mathbf{x},\mathbf{y}) = (u(\mathbf{x}) - u(\mathbf{y}))^2 |\mathbf{x}-\mathbf{y}|^{-n-2s} \mathbb{1}\{|\mathbf{x}-\mathbf{y}| \le \delta\}$, consider:

$$\begin{aligned}
\int_{\mathbb{R}^n}\int_{\mathbb{R}^n} G\, d\mathbf{y} d\mathbf{x} &= \int_{\Omega\cup\Omega_I}\int_{\mathbb{R}^n} G\, d\mathbf{y} d\mathbf{x} + \int_{\mathbb{R}^n\setminus\Omega\cup\Omega_I}\int_{\mathbb{R}^n} G\, d\mathbf{y} d\mathbf{x} \\
&= \int_{\Omega\cup\Omega_I}\int_{\Omega\cup\Omega_I} G\, d\mathbf{y} d\mathbf{x} + \int_{\Omega\cup\Omega_I}\int_{\mathbb{R}^n\setminus\Omega\cup\Omega_I} G\, d\mathbf{y} d\mathbf{x} + \int_{\mathbb{R}^n\setminus\Omega\cup\Omega_I}\int_{\mathbb{R}^n} G\, d\mathbf{y} d\mathbf{x} \\
&= \int_{\Omega\cup\Omega_I}\int_{\Omega\cup\Omega_I} G\, d\mathbf{y} d\mathbf{x} + \int_{\Omega\cup\Omega_I}\int_{\mathbb{R}^n\setminus\Omega\cup\Omega_I} G\, d\mathbf{y} d\mathbf{x} \\
&\quad + \int_{\mathbb{R}^n\setminus\Omega\cup\Omega_I}\int_{\Omega\cup\Omega_I} G\, d\mathbf{y} d\mathbf{x} + \int_{\mathbb{R}^n\setminus\Omega\cup\Omega_I}\int_{\mathbb{R}^n\setminus\Omega\cup\Omega_I} G\, d\mathbf{y} d\mathbf{x}.
\end{aligned}$$

Here, the last term vanishes because $u \in H^s_\Omega(\mathbb{R}^n)$. For the same reason, and by definition of G, we have that

$$\begin{aligned}
\int_{\mathbb{R}^n}\int_{\mathbb{R}^n} G\, d\mathbf{y} d\mathbf{x} &= E_{tr}(u;\delta) + \int_{\Omega\cup\Omega_I} u^2(\mathbf{x}) \int_{\mathbb{R}^n\setminus\Omega\cup\Omega_I} |\mathbf{x}-\mathbf{y}|^{-n-2s} \mathbb{1}\{|\mathbf{x}-\mathbf{y}| \le \delta\} d\mathbf{x} d\mathbf{y} \\
&\quad + \int_{\Omega\cup\Omega_I} u^2(\mathbf{y}) \int_{\mathbb{R}^n\setminus\Omega\cup\Omega_I} |\mathbf{x}-\mathbf{y}|^{-n-2s} \mathbb{1}\{|\mathbf{x}-\mathbf{y}| \le \delta\} d\mathbf{x} d\mathbf{y} \\
&= E_{tr}(u;\delta) + \int_{\Omega} u^2(\mathbf{x}) \int_{\mathbb{R}^n\setminus\Omega\cup\Omega_I} |\mathbf{x}-\mathbf{y}|^{-n-2s} \mathbb{1}\{|\mathbf{x}-\mathbf{y}| \le \delta\} d\mathbf{x} d\mathbf{y} \\
&\quad + \int_{\Omega} u^2(\mathbf{y}) \int_{\mathbb{R}^n\setminus\Omega\cup\Omega_I} |\mathbf{x}-\mathbf{y}|^{-n-2s} \mathbb{1}\{|\mathbf{x}-\mathbf{y}| \le \delta\} d\mathbf{x} d\mathbf{y} \\
&= E_{tr}(u;\delta).
\end{aligned}$$

Where the last equality follows from that fact that for $\mathbf{x} \in \Omega$ and $\mathbf{y} \in \mathbb{R}^n\setminus\Omega\cup\Omega_I$, $|\mathbf{x}-\mathbf{y}| > \delta$; i.e. the indicator function is zero. □

The next lemma shows that for every λ there exists some value of the truncation parameter, $\overline{\delta}$ for which the tempered energy is bounded by the truncated energy associated with $\overline{\delta}$.

LEMMA 5.3. *For $\lambda > 0$, there exists a $\overline{\delta} > 0$ independent of u such that*

$$E_{tem}(u;\lambda) \le E_{tr}(u;\overline{\delta}), \quad \forall\, u \in H^s_\Omega(\mathbb{R}^n).$$

Proof. First, note that for any λ, $E_{tem}(u;\lambda) \le E_{tr}(u;\infty)$. Next, we show that there exists $\overline{\delta}$ such that $E_{tr}(u;\infty) \le 2E_{tr}(u;\overline{\delta})$. This result is equivalent to

$$E_{tr}(u;\infty) - E_{tr}(u;\overline{\delta}) \le \frac{1}{2} E_{tr}(u;\infty).$$

Using Lemma 5.2,

$$\begin{aligned}
&E_{tr}(u;\infty) - E_{tr}(u;\delta) \\
&\qquad = \int_{\mathbb{R}^n}\int_{\mathbb{R}^n} (u(\mathbf{x})-u(\mathbf{y}))^2 |\mathbf{x}-\mathbf{y}|^{-n-2s}(1-\mathbb{1}\{|\mathbf{x}-\mathbf{y}|\le\delta\})d\mathbf{x}d\mathbf{y} \\
&\qquad \le \delta^{-n-2s}\int_{\mathbb{R}^n}\int_{\mathbb{R}^n} (u(\mathbf{x})-u(\mathbf{y}))^2 (1-\mathbb{1}\{|\mathbf{x}-\mathbf{y}|\le\delta\})d\mathbf{x}d\mathbf{y} \\
&\qquad \le \delta^{-n-2s}\int_{\mathbb{R}^n}\int_{\mathbb{R}^n} (u(\mathbf{x})-u(\mathbf{y}))^2 d\mathbf{x}d\mathbf{y}.
\end{aligned}$$

Since $u \equiv 0$ on $\mathbb{R}^n \setminus \Omega$, we have

$$\delta^{-n-2s}\int_{\mathbb{R}^n}\int_{\mathbb{R}^n} (u(\mathbf{x})-u(\mathbf{y}))^2 d\mathbf{x}d\mathbf{y} \le C\delta^{-n-2s}\|u\|^2_{L^2(\Omega)}.$$

By invoking Lemma 4.3 of [4], we have

$$C\delta^{-n-2s}\|u\|^2_{L^2(\Omega)} \le C'\delta^{-n-2s}E_{tr}(u;\infty).$$

We conclude the proof by choosing $\delta = \overline{\delta}$ such that $C'\delta^{-n-2s} \le 1/2$. ∎

We recall the following result from [2] that state that all truncated energies are equivalent.

THEOREM 5.4. *For any $\delta, \delta' > 0$, there exist constants C_1 and C_2 such that*

$$C_1 E_{tr}(u;\delta) \le E_{tr}(u;\delta') \le C_2 E_{tr}(u;\delta).$$

Combining Lemma 5.3 and Theorem 5.4 we obtain the following estimate for the right-hand side of (5.3).

THEOREM 5.5. *Given $\lambda > 0$,*

$$E_{tem}(u;\lambda) \le \overline{A}E_{tr}(u;\delta), \quad \forall\, u \in H^s_\Omega(\mathbb{R}^n),$$

where the positive constant $\overline{A}$ depends on Ω and is independent of u.

6. Conclusions. We discussed the consistency of the unified nonlocal Laplacian operator introduced in [3] with the tempered and truncated fractional Laplacian operators via the equivalence kernel. With several numerical tests, we illustrated our theoretical results, confirming that the composition of tempered fractional divergence and gradient yields the tempered fractional Laplacian, whereas the composition of truncated fractional divergence and gradient does not yield, as one might expect, a truncated fractional Laplacian operator.

With the purpose of identifying an operator that is equivalent to the tempered fractional Laplacian, but computationally cheaper, we investigated the relationship between the tempered fractional and truncated fractional energy norms and showed that for a fixed tempered parameter λ, the tempered energy is equivalent to any truncated energy. This result represents a step forward towards the identification of computationally cheap alternatives to fractional operators whose integration domain spans the whole space $\mathbb{R}^n$.

7. Acknowledgments. This work was supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research under the Collaboratory on Mathematics and Physics-Informed Learning Machines for Multiscale and Multiphysics Problems (PhILMs) project.

REFERENCES

[1] D. A. Benson, S. W. Wheatcraft, and M. M. Meerschaert, *Application of a fractional advection-dispersion equation*, Water Resources Research, 36 (2000), pp. 1403–1412.

[2] O. Burkovska and M. Gunzburger, *Affine approximation of parametrized kernels and model order reduction for nonlocal and fractional laplace models*, SIAM Journal on Numerical Analysis, 58 (2020), pp. 1469–1494.

[3] M. D'Elia, M. Gulian, H. Olson, and G. Karniadakis, *A unified theory of fractional, nonlocal, and weighted nonlocal vector calculus*, (2020). ArXiv preprint 2005.07686.

[4] Q. Du, M. Gunzburger, R. B. Lehoucq, and K. Zhou, *Analysis and approximation of nonlocal diffusion problems with volume constraints*, SIAM Review, 54 (2012), pp. 667–696.

[5] A. Katiyar, S. Agrawal, H. Ouchi, P. Seleson, J. T. Foster, and M. M. Sharma, *A general peridynamics model for multiphase transport of non-Newtonian compressible fluids in porous media*, Journal of Computational Physics, (2019). In press.

[6] A. Katiyar, J. T. Foster, H. Ouchi, and M. M. Sharma, *A peridynamic formulation of pressure driven convective fluid transport in porous media*, Journal of Computational Physics, 261 (2014), pp. 209–229.

[7] P. C. D. Leoni, T. A. Zaki, G. Karniadakis, and C. Meneveau, *Two-point stress-strain rate correlation structure and non-local eddy viscosity in turbulent flows*. ArXiv preprint 2006.02280., 2020.

[8] G. Pang, M. D'Elia, M. Parks, and G. E. Karniadakis, *nPINNs: nonlocal Physics-Informed Neural Networks for a parametrized nonlocal universal Laplacian operator. Algorithms and Applications*, Journal of Computational Physics, to appear, (2020).

[9] F. Sabzikar, M. M. Meerschaert, and J. Chen, *Tempered fractional calculus*, J. Comput. Phys., 293 (2015), pp. 14–28.

[10] A. A. Schekochihin, S. C. Cowley, and T. A. Yousef, *MHD turbulence: Nonlocal, anisotropic, nonuniversal?*, in IUTAM Symposium on computational physics and new perspectives in turbulence, Springer, Dordrecht, 2008, pp. 347–354.

[11] R. Schumer, D. Benson, M. Meerschaert, and S. Wheatcraft, *Eulerian derivation of the fractional advection-dispersion equation*, Journal of Contaminant Hydrology, 48 (2001), pp. 69–88.

[12] R. Schumer, D. A. Benson, M. M. Meerschaert, and B. Baeumer, *Multiscaling fractional advection-dispersion equations and their solutions*, Water Resources Research, 39 (2003), pp. 1022–1032.

[13] Y. Wei, Y. Kang, W. Yin, and Y. Wang, *Generalization of the gradient method with fractional order gradient direction*, arXiv:1901.05294v2, (2020).

AMG FOR MIXED FINITE ELEMENT REPRESENTATIONS OF SYSTEMS OF PDES

ALEXEY VORONIN*, RAYMOND TUMINARO†, LUKE OLSON‡, AND SCOTT MACLACHLAN§

Abstract. We consider algebraic multigrid (AMG) preconditioners for systems of partial differential equations (PDEs) discretized with mixed finite elements, where the degrees of freedom (DoFs) corresponding to different quantities are not necessarily co-located at the mesh points. We specifically focus on applications such as the linearized steady-state Navier-Stokes equations, which give rise to saddle-point systems. Within algebraic multigrid solvers, coarse-level discretization operators are automatically and algebraically created by the solver. However, it is not generally well understood how to guarantee the stability of these automatically generated operators for different discretizations and whether the stability is strictly necessary to achieve satisfactory convergence. The focus of this work is on developing practical and theoretical guidance into the different multigrid choices for this problem. We experimentally explore several AMG variants, which highlight significantly different convergence histories. Then we revisit some fundamental concepts in multigrid theory with a goal of adapting the theory to indefinite Stokes systems. It is anticipated that this will lead to a better understanding of the different AMG variants and their associated convergence behavior.

1. Introduction. This study focuses on the numerical solution of the Stokes equations, which are used to simulate incompressible viscous flow. The Stokes equations can be seen as a linearization of the more general Navier-Stokes equations, which are at the heart of many scientific and engineering applications. The key challenge in this work is that a discretization of the Stokes equations leads to an indefinite and ill-conditioned saddle-point problem [5].

For positive-definite linear systems, multigrid methods have been shown to be fast and effective solvers due to their arithmetic efficiency and grid-independent convergence [4, 15].The rapid convergence rate of multigrid relies on the complementary nature of its two main components: relaxation and coarse-grid correction. Relaxation reduces oscillatory error components, while the coarse-grid correction addresses smooth error by "solving" a low-resolution version of the equations. This idea is then applied recursively, forming a hierarchy of grids and grid transfers.

When using geometric multigrid (GMG) solvers, the hierarchy is simply composed of a sequence of coarser discretizations of the original problem. However, the formation of geometric inter-grid transfers is often challenging for applications that require complex meshes. Algebraic multigrid (AMG) solvers are an attractive alternative in this situation because the grid hierarchy and grid transfers are constructed automatically using only the matrix and graph properties [4, 14, 15].

This paper investigates the application of AMG to sparse linear systems that arise from discretizations of the Stokes equations using mixed finite elements. In the case of the mixed finite-element discretization, the unknowns pertaining to different physical quantities are not always co-located at the same mesh point. For co-located discretizations, it is common to group DoFs at each mesh node to define a vertex of a nodal matrix graph that is then coarsened to construct a coarse grid [7]. This AMG approach for dealing with PDE systems where unknowns are co-located is not applicable in the general or mixed FE case where DoFs are not co-located. Developing AMG hierarchies to address these general mixed finite-element cases remains an open problem. In this paper, we examine the convergence behavior of several multigrid variants for a non-co-located discretization of the Stokes equations. In addition, we consider several theoretical observations to understand the pitfalls of applying AMG to indefinite Stokes systems.

*University of Illinois Urbana Champaign, voronin2@illinois.edu
†Sandia National Laboratories, rstumin@sandia.gov
‡University of Illinois Urbana Champaign, lukeo@illinois.edu
§Memorial University of Newfoundland, smaclachlan@mun.ca

2. Problem Formulation. We develop multigrid approaches to solve the Stokes equations, which model incompressible viscous fluid flow. The Stokes equations can be expressed as

$$\begin{aligned} -\nabla^2 \mathbf{u} + \nabla p &= f \quad \text{in} \quad \Omega \\ -\nabla \cdot \mathbf{u} &= 0 \quad \text{in} \quad \Omega \end{aligned} \tag{2.1}$$

for some arbitrary domain $\Omega \in \mathbb{R}^2$, where $\mathbf{u}$ is a vector-valued function representing velocity, p is a scalar pressure function, and f is a forcing term. The above equations are accompanied by a set of boundary conditions on $\partial\Omega = \Gamma_D \cup \Gamma_N$ such that

$$\mathbf{u} = \mathbf{w} \text{ on } \Gamma_D, \quad \frac{\delta \mathbf{u}}{\delta \mathbf{n}} - \mathbf{n}p = \mathbf{s} \text{ on } \Gamma_N \tag{2.2}$$

where $\mathbf{n}$ is an outward-pointing unit-length normal vector to the boundary and $\delta\mathbf{u}/\delta\mathbf{n}$ is the directional derivative in the normal direction. The boundary condition terms $\mathbf{w}$ and $\mathbf{s}$ are given.

The corresponding weak formulation of the Stokes equations is the following: Find $\mathbf{u} \in \mathbf{H}^1(\Omega)$ and $p \in L_2(\Omega)$ such that

$$\begin{aligned} (\nabla \mathbf{u}, \nabla \mathbf{v}) - (\nabla \cdot \mathbf{v}, p) &= (f, \mathbf{v}) \text{ for all } v \in H_0^1(\Omega)^d \\ -(\nabla \cdot \mathbf{u}, q) \qquad\qquad &= 0 \qquad \text{for all } q \in L_0^2(\Omega) \end{aligned} \tag{2.3}$$

Discretizing the weak formulation using two suitably chosen finite-dimensional subspaces, $V \in H_0^1$ and $W \in L^2$, leads to a general algebraic system given in the following form:

$$\mathcal{A}x = \begin{bmatrix} M & B^T \\ B & -C \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ p \end{bmatrix} = \begin{bmatrix} f \\ 0 \end{bmatrix} \tag{2.4}$$

where $C = 0$.

In order to guarantee the discrete solvability of the above system, the finite-element spaces V and W need to satisfy the so-called *inf-sup* stability condition. Many seemingly appropriate element space pairings result in an unstable system, for example, $\mathbb{Q}_1/Q_1$ where a bilinear approximation space is used for both velocity and pressure. Unstable discretizations typically yield unphysical oscillatory solutions even when the boundary conditions are smooth. The stability of the $\mathbb{Q}_1/Q_1$ discretization can be recovered by replacing $C = 0$ with an appropriately scaled bilinear stiffness matrix associated with the Neumann boundary condition [5].

For the purpose of this paper, we have chosen $\mathbb{Q}_2/Q_1$ discretization, which represents biquadratic approximation for the velocity space and the bilinear approximation space for pressure. The $\mathbb{Q}_2/Q_1$ element discretization satisfies the *inf-sup* condition and therefore is stable. The DoF locations for the $\mathbb{Q}_2/Q_1$ element are depicted in Figure 2.1. For a detailed discussion of the above-mentioned discretization and their stability proofs the reader is referred to [5].

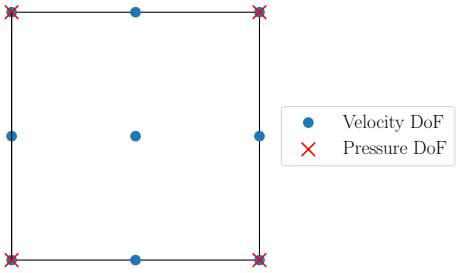


Fig. 2.1: $\mathbb{Q}_2/Q_1$ Finite Element Pair

3. Multigrid Methods. The effectiveness of multigrid (MG) methods comes from the decomposition of grid functions into two energy-orthogonal subspaces: one for high-energy (or oscillatory) components and another for low-energy (or smooth) components. The oscillatory components should be effectively reduced via some simple iterative scheme, usually referred to as relaxation. The residual equation is then projected onto a low-energy subspace, also known as the coarse grid, where an approximate solution to the coarse-grid problem can be computed more easily. The coarse solution is then interpolated to the fine grid and is used to correct the fine-level solution. The multilevel version of this is called the multigrid V-cycle, which is described in the Algorithm 1.

Algorithm 1 MG V-cycle

Input: x_0, fine-level initial guess
  b_0, right-hand side
  $A_0, ..., A_{l_{max}-1}$; PDE discretization operator for each level
  $P_1, ..., P_{l_{max}-1}$; Interpolation for each coarse grid
Output: x_0, fine-level approximation after one V-cycle

for $l = 0, ..., l_{max} - 1$ **do**
  $relax(A_l, x_l, b_l)$ // Pre-relaxation
  $r_{l+1} = P_{l+1}^T(b_l - A_l x_l)$ // Project residual onto coarse grid
  $x_{l+1} = 0$
end for
$x_{l_{max}-1} = solve(A_{l_{max}-1}, r_{l_{max}-1})$ // Direct solve on coarsest level
for $l = l_{max} - 2, ..., 0$ **do**
  $x_l = x_l + P_{l+1} x_{l+1}$ // Interpolate and correct
  $relax(A_l, x_l, b_l)$ // Post-relaxation
end for

To set up a multigrid solver we need to define A_l, P_l, and the *relax* function. In our setting, for $l > 1$, the A_l are formed via Galerkin coarse-grid approximation $A_l = P_l^T A_{l-1} P_l^T$. The construction of the interpolation operators P_l is discussed in Sections 3.1 - 3.3. In this paper, we utilize a coupled block relaxation method for incompressible flow problems called Brass-Sarazin relaxation, which is described in Section 3.4.

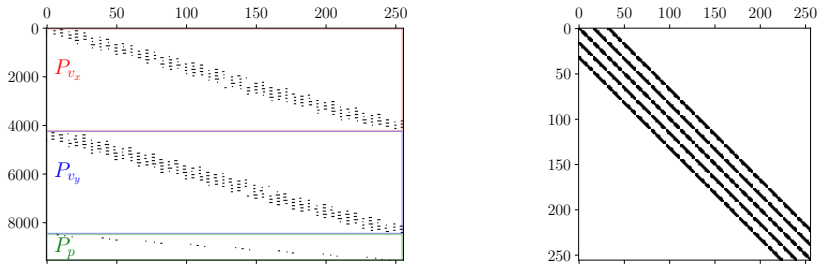
3.1. Coarsening. If grids are structured, then coarsening of grids, as in GMG, is typically done by halving the number of degrees of freedom in each dimension, thereby doubling the element edge length on the coarse grid. The interpolation operator is then

defined as a weighted sum of the neighboring DoFs corresponding to piecewise polynomial interpolation. This leads to a sequence of coarser discretizations of the original problem and allows all grids in the hierarchy to preserve the notion of the finite element.

In contrast, in an AMG method, the coarse grid is formed automatically based on the graph of the fine-level operator, oftentimes without any geometric grid data. This is an attractive and oftentimes necessary approach when dealing with unstructured mesh problems.

There exist several different AMG coarsening approaches. This paper primarily discusses *Smoothed Aggregation* based AMG (SA). AMG setup first determines the coarse grid then defines an appropriate interpolation operator. In SA, each coarse-grid DoF is defined as a collection of fine-level DoFs (also known as an aggregate), where the grouping/aggregation of the fine-level DoFs is based on the undirected adjacency graph of the matrix. The aggregate information is then used to define an interpolation operator that accurately interpolates error that is not effectively reduced by relaxation.

3.2. Monolithic Interpolation. We can create a multilevel solver by providing the assembled matrix $\mathcal{A}$ (2.4) without any other information to Smoothed Aggregation (SA) solver (such as that found in PyAMG [11]). The problem with this approach is that SA does not differentiate between velocity and pressure DoFs in the aggregation process, leading to aggregates with a mix of different unknowns: velocity and pressure DoFs. This is problematic because the interpolation operator, as shown in the Figure 3.1(a), mixes different DoF types (v_x, v_y, p). That is, a column of the interpolation matrix will generally have column entries corresponding to the different DoF types. The Galerkin coarse-grid approximation using these grid-transfer operators results in a coarse-grid discretization matrix (Figure 3.1(b)) with a non-zero lower diagonal block, thus not having the saddle-point structure.



(a) Interpolation Operator Sparsity Pattern (b) Coarse-grid Sparsity Pattern

Fig. 3.1: Naive MG Sparsity Patterns. Each marker corresponds to a nonzero entry in the matrix.

3.3. Component-wise interpolation. In order to preserve the saddle-point structure of the matrix on the coarse grids, we can construct block diagonal interpolation operators, P, where all components of the solution interpolate only from the coarse DoFs of the same components: velocity (P_v) and pressure (P_p). As demonstrated in Equation (3.1), the saddle point structure of the coarse-grid operator, $\mathcal{A}_c$, is preserved when the coarse-grid operator

is formed with a block diagonal interpolation operator, P.

$$\mathcal{A}_c = P^T \mathcal{A} P = \begin{bmatrix} P_v^T & \\ & P_p^T \end{bmatrix} \begin{bmatrix} M & B^T \\ B & 0 \end{bmatrix} \begin{bmatrix} P_v & \\ & P_p \end{bmatrix} = \begin{bmatrix} P_v^T M P_v & P_v^T B^T P_p \\ P_p^T B P_v & 0 \end{bmatrix} \tag{3.1}$$

3.3.1. Geometric Interpolation. The geometric interpolation operators for the Stokes system described in Section 2 are defined using the piecewise basis functions associated with a $\mathbb{Q}_2/Q_1$ discretization of the coarse grid: a biquadratic basis (P_v) for the velocity elements and a bilinear basis (P_p) for the pressure. Preserving the same finite element discretization on all grids guarantees the *inf-sup* stability of the coarse-grid operators.

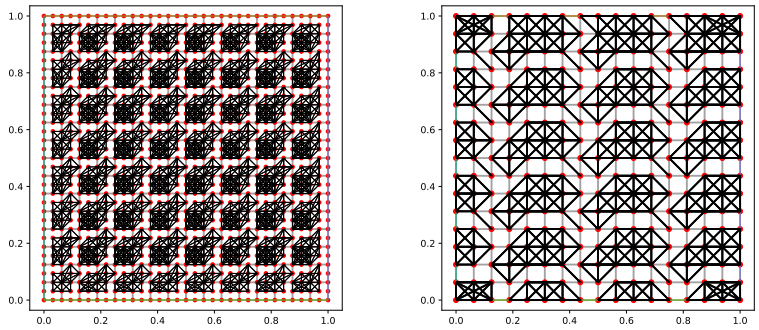
3.3.2. Separate SA Interpolation. The interpolation operators in AMG can also be constructed to have a block-diagonal structure. To do so, we exploit the block-structure in $\mathcal{A}$, forming a smoothed aggregation hierarchy for each component of velocity and pressure independently. Although this yields a saddle-point structure, the coarse-grid operator is not guaranteed to satisfy the same *inf-sup* stability condition that the fine-grid discretization has.

We construct the velocity interpolation operator by applying SA to the vector-Laplacian block, M. As the (2,2)-block of (2.4) is zero, it obviously cannot be used in conjunction with an SA procedure to generate an interpolation operator. Instead, some type of auxiliary operator needs to be constructed. The operator BB^T corresponds to pressure Poisson operator and so is a potential candidate for this auxiliary matrix. The only technical problem is that for a $\mathbb{Q}_2/Q_1$ discretization, BB^T has a somewhat denser stencil (with non-nearest neighbor entries) than is desired for the AMG coarsening procedures found in most SA packages. To correct this, we adopt the same approach described in [12]. First, we remove off-diagonal entries of $Z = BB^T$ that satisfy $|Z| \leq \tau\sqrt{|Z_{ii}Z_{jj}|}$, where τ is an appropriately chosen constant, and lump them onto the diagonal so that the row sum is preserved. For more details on this technique, the reader is referred to [12].

Since the correlation between pressure and velocity aggregation patterns is not enforced, the coarse-grid velocity basis and pressure functions are also uncorrelated. As a result, the relative ratio of pressure DoFs on the coarse grid is closer to one-to-one, which resembles more the unstable $\mathbb{Q}_1/Q_1$ discretization rather than $\mathbb{Q}_2/Q_1$ discretization that we chose. The structure and the relative positioning of the pressure and velocity aggregates is depicted in Figures 3.2-3.3 for two different aggregation strategies: standard and Lloyd Aggregation. Lloyd aggregation is a coarsening approach that utilizes k-means clustering for graphs in order to distribute/seed aggregates centers in a way that minimizes the average distance of a given point to the nearest center aggregate center [1]. In the context of our research, it was used to examine the effect of aggregation techniques on the convergence.

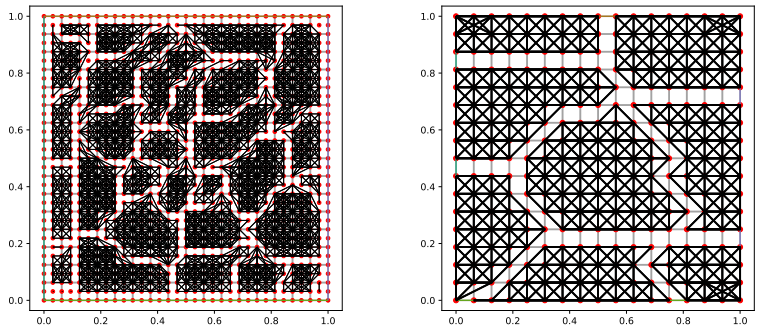
3.3.3. Co-aggregation. While the separate SA approach preserves the saddle-point structure of the matrix, the 4:1 ratio between velocity DoFs and pressure DoFs' is not maintained on coarser grids. This brings forth concerns regarding coarse-grid stability. The co-aggregation approach is motivated by trying to keep the ratio of DoFs on the coarse grid similar to that of the fine-level matrix. This 'correct' ratio would still not guarantee the stability of the coarse-grid operators but is perhaps a step in the right direction.

We have identified but not tested several approaches that would help us preserve the desired ratio. One such approach we refer to as tiling. During the first pass of the aggregation phase, we would group the velocity DoFs into 2×2 bricks, which we call *supernodes*. For the second pass, the *supernodes* are aggregated. These *supernode* aggregates are then post-processed so that each *supernode* aggregate defines one coarse-pressure DoF and four coarse-velocity DoFs. The same idea would hold in 3D for an 8:1 ratio. The construction of the interpolation operator using the coarsened data is still largely an open question to us.



(a) Coarse and Fine Velocity DoFs (b) Coarse and Fine Pressure DoFs

Fig. 3.2: Standard Aggregation. The plots depict velocity/pressure DoFs (red), mesh edges (light grey), and aggregates (collection of black lines connecting the red dots).



(a) Coarse and Fine Velocity DoFs (b) Coarse and Fine Pressure DoFs

Fig. 3.3: Lloyd Aggregation. The plots depict velocity/pressure DoFs (red), mesh edges (light grey), and aggregates (collection of black lines connecting the red dots).

3.4. Relaxation. Classical relaxation methods such as Jacobi and Gauss-Seidel are not well-defined for saddle-point systems due to the large zero block and the indefinite system. As a result, we turn to Braess-Sarazin relaxation, which exploits the block structure of the system to produce a smoothing property [2].

Braess-Sarazin relaxation is based on the block LU factorization of the discretized system, $\mathcal{A}$, in Equation (2.4):

$$\begin{bmatrix} \alpha\hat{M} & B^T \\ B & 0 \end{bmatrix} \begin{bmatrix} \mathbf{ffiu} \\ \delta p \end{bmatrix} = \begin{bmatrix} \alpha\hat{M} & 0 \\ B & S \end{bmatrix} \begin{bmatrix} I & \frac{1}{\alpha}\hat{M}^{-1}B^T \\ 0 & I \end{bmatrix} \begin{bmatrix} \mathbf{ffiu} \\ \delta p \end{bmatrix} = \begin{bmatrix} r_u \\ r_p \end{bmatrix} \tag{3.2}$$

where $\hat{M}$ is a preconditioner for M, $S = \frac{1}{\alpha}B\hat{M}^{-1}B^T$ and α is a relaxation parameter.

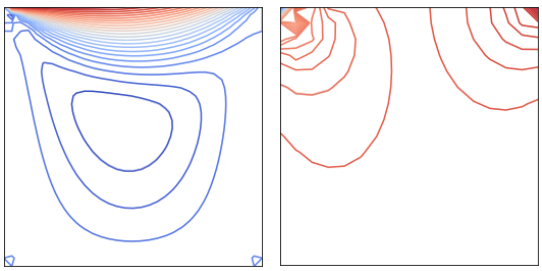
The robustness of this algorithm relies on the ease of computability of $\hat{M}^{-1}$. In our case, $D = \hat{M}$ =diag(M), which makes the inversion trivial and reduces the overall algorithm to two sequential steps:

$$\begin{aligned} S\delta p &= \frac{1}{\alpha} B D^{-1} r_u - r_p \\ \delta \mathbf{u} &= \frac{1}{\alpha} D^{-1}(r_u - B^T \delta p) \end{aligned} \tag{3.3}$$

where $\delta \mathbf{u}$ and δp correspond to the velocity and pressure residual corrections. The Schur complement equation is approximately solved using Jacobi relaxation.

4. Experiments. In this section, we present results for the Stokes problem using multigrid solvers presented in the previous sections. The solvers are applied to the traditional lid-driven-cavity-flow problem on a square mesh ($\Omega = [0,1]^2$), where the boundary condition (Equations (2.2)) parameters $\mathbf{w}$ and $\mathbf{s}$ are given. The velocity, $\mathbf{w}$, is non-zero only on the top side of a rectangular domain and is prescribed by the regularized cavity model $\{y = 1, : 0 \leq x \leq 1 \mid u_x = 1 - x^4\}$. The $\mathbf{s}$ is set to be zero everywhere along the boundary in order to satisfy the computability condition of the boundary data.

The algebraic systems were generated using Firedrake system [13] and the proposed algorithms were constructed with the help of PyAMG library [11]. A typical solution for a 10×10 mesh is depicted in Figure 4.1.



(a) Velocity Solution (b) Pressure Solution

Fig. 4.1: Equally distributed streamlines for the solution to Stokes lid-cavity problem.

We used left preconditioned GMRES as an iterative method with stopping criteria of 1e-9 relative residual tolerance or 100 maximum iterations, whichever is reached first. Each application of the multigrid preconditioner entails a single V-cycle. In our analysis of the convergence, we are interested in seeing a relatively small number of iterations to the convergence, and h-independence - the number of GMRES iterations doesn't grow as the mesh is refined.

In all of the numerical results (unless otherwise stated), we use two sweeps of Braess-Sarazin (BS) for the pre-/post- relaxation. Each BS relaxation application involves three iterations of Jacobi relaxation to approximate the solution of the Schur complement system. The BS damping parameter, α, was set to 1.01 and the Jacobi damping parameter was set to $\omega_J = 1$ for GMG [6]. In the case of AMG, we find that $\omega_J = 1/2$ results in better convergence.

4.1. GMG. The convergence results for the geometric multigrid preconditioned GMRES solver are shown in Figure 4.2. The results depict MG convergence for varying mesh

sizes and the number of multigrid levels. The desired h-independent convergence is achieved, which causes some of the lines to overlap.

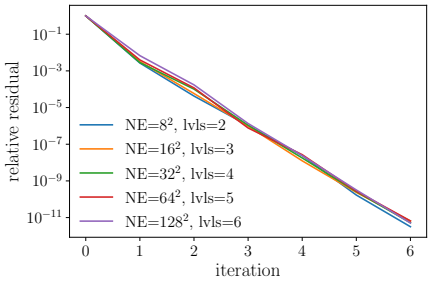


Fig. 4.2: GMG Preconditioned GMRES Convergence Results

4.2. Naive AMG. As mentioned previously, the default PyAMG's SA setup constructs a series of coarse grids that do not have a saddle-point structure. Hence, Braess-Sarazin relaxation is not well defined for those grids. Luckily, PyAMG provides a default relaxation procedure - point-wise Gauss-Seidel, which we will use for all levels of the hierarchy. PyAMG's implementation of the relaxation ignores rows that have zero on the diagonal, which allows the relaxation not to break down on the fine level. While the coarse grid does not have the zero-block like the fine-level operator, the coarse-grid DoFs represent a mixture of different fine-level unknowns for which the smoothing property is uncertain. The Naive AMG's convergence flatlines after several iterations (Figure 4.3).

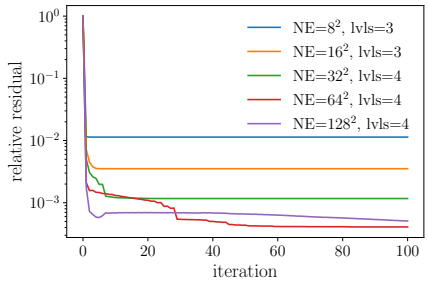
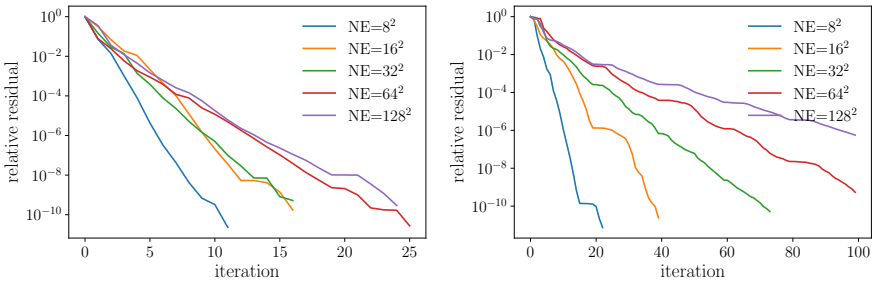


Fig. 4.3: Naive AMG: Convergence Results

4.3. Separate AMG. In the case of Separate AMG, the multigrid hierarchies for pressure and velocity interpolations are formed independently of each other. The parameters passed to the SA setup phase for each field are shown in Table 4.1. Both for pressure and velocity fields, we used symmetric strength of connection between the DoFs, and two different types of DoF aggregation approaches. For more information about these parameters please refer to the PyAMG library [11]. Since the separate AMG approach is sensitive to the aggregation techniques chosen, we show the convergence results for both 'Lloyd' and 'standard' aggregation (see Figure 4.4).

Table 4.1: Smoothed Aggregation Parameters

Field	**Aggregation Matrix**	**Smooth**
Velocity	M	Jacobi($\omega = 4/3$)
Pressure	$Z = BB^T$ $\hat{Z}$ =drop_small_entries(Z)	energy



(a) Standard Aggregation (b) Lloyd Aggregation

Fig. 4.4: Separate AMG: Convergence Results

Using the AMG methods as stationary iterations (not shown here) produces significantly worse convergence histories. Overall, the total number of iteration is not that high for the meshes that we have been able to investigate. However, there is a general growth in the number of iterations as well as a sensitivity to the aggregation scheme, both of which are cause for concern. In order to identify what is going wrong the AMG v-cycle, we turn to the convergence analysis.

Table 4.2 illustrates the ratio of x velocity DoFs to pressure DoFs, demonstrating how it can stray far from the ratio on the finest grid. This does not necessarily imply a stability issue but might indicate an area of concern

Table 4.2: Ratio of velocity (x-component) to pressure DoFs

MG level	**Standard Aggregation**	**Lloyd Aggregation**
0	3.97	3.97
1	2.22	3.97
2	1.67	4.21
3	1.28	1.0

4.4. Convergence Analysis. The main goal for our future work is to better understand the role the coarse-grid stability plays in the multigrid convergence, especially in the $\mathbb{Q}_2/Q_1$ case.

To understand the stability issues, we have begun an investigation into a two-level MG hierarchy where the fine-level grid is a $\mathbb{Q}_2/Q_1$ discretization and the coarse-level grid is $\mathbb{Q}_1/Q_1$, where bi-linear interpolation is used to transfer corrections between grids. That is, we have purposely chosen an unstable coarse-grid discretization. Interestingly, our exper-

iments show that a two-level MG solver converges without issue any issues in a relatively mesh independent fashion, even though the coarse grid is unstable. This may lead one to believe that coarse-operator stability is unimportant. However, multilevel (greater than 2) GMG results using $\mathbb{Q}_1/Q_1$ coarse grids yield more problematic results from a convergence perspective.

To understand this example as well as the results presented in Section 4.3, we want to consult the two main principles underlying MG theory: smoothing and approximation properties. The smoothing property describes a relaxation method's ability to cope with a high-frequency error. The approximation property measures how well the relaxed error can be represented on the coarse grid. These two properties are used to prove that the reduction in error by a factor that is independent of the size of the grid, which is also known as h-independence.

Techniques for establishing these properties were studied by several authors in the early 1980s, applied to both geometric and algebraic multigrid [10, 3, 9]. Convergence analysis for AMG is nicely summarized in the seminal paper of Ruge and Stüben [14], at least for the case of symmetric and positive-definite systems. Since then more practical, although less sharp, convergence bounds have been developed and applied to diagnose AMG convergence for SPD systems [8]. However, due to the indefiniteness of the Stokes' system, this analysis is not directly applicable.

Some research has been published on extending the above-mentioned principles to GMG for the saddle point systems [16]. The smoothing and approximation properties in those cases depend on norms scaled by the mass matrices and the element edge length parameter, h, which are not always well defined for the coarse grids of AMG hierarchies. Hence, in order to adapt the smoothing and approximation principles to AMG for saddle-point systems, we first need to define the appropriate norm.

4.5. Conclusion. AMG preconditioners for systems of PDEs discretized with mixed finite elements are difficult to construct, especially where the degrees of freedom (DoFs) corresponding to two different quantities are not co-located at the mesh points. We have constructed AMG hierarchies that preserve the saddle-point nature of the original problem, which resulted in reasonable although not h-independent, convergence for some choice of parameters. In order to better understand these results, we turn to MG theory for the future direction of the research.

REFERENCES

[1] W. N. Bell, *Algebraic multigrid for discrete differential forms*, (2008).
[2] D. Braess and R. Sarazin, *An efficient smoother for the stokes problem*, Applied Numerical Mathematics, 23 (1997), pp. 3–19.
[3] A. Brandt, *Algebraic multigrid theory: The symmetric case*, Applied mathematics and computation, 19 (1986), pp. 23–56.
[4] W. Briggs, V. Henson, and S. F. McCormick, *A multigrid tutorial, 2nd*, Edition. SIAM, Philadelphia, PA, (2000).
[5] H. C. Elman, D. J. Silvester, and A. J. Wathen, *Finite elements and fast iterative solvers: with applications in incompressible fluid dynamics*, Oxford University Press, USA, 2014.
[6] Y. He and S. P. MacLachlan, *Local fourier analysis for mixed finite-element methods for the stokes equations*, Journal of Computational and Applied Mathematics, 357 (2019), pp. 161–183.
[7] A. Janka, *Smoothed aggregation multigrid for a stokes problem*, Computing and Visualization in Science, 11 (2008), pp. 169–180.
[8] S. P. MacLachlan and L. N. Olson, *Theoretical bounds for algebraic multigrid performance: review and analysis*, Numerical Linear Algebra with Applications, 21 (2014), pp. 194–220.
[9] J. Mandel, *Algebraic study of multigrid methods for symmetric, definite problems*, Applied mathematics and computation, 25 (1988), pp. 39–56.

[10] S. McCormick, *Multigrid methods for variational problems: general theory for the v-cycle*, SIAM Journal on Numerical Analysis, 22 (1985), pp. 634–643.
[11] L. N. Olson and J. B. Schroder, *PyAMG: Algebraic multigrid solvers in Python v4.0*, 2018. Release 4.0.
[12] A. Prokopenko and R. S. Tuminaro, *An algebraic multigrid method for q 2- q 1 mixed discretizations of the navier–stokes equations*, Numerical Linear Algebra with Applications, 24 (2017), p. e2109.
[13] F. Rathgeber, D. A. Ham, L. Mitchell, M. Lange, F. Luporini, A. T. T. McRae, G.-T. Bercea, G. R. Markall, and P. H. J. Kelly, *Firedrake: automating the finite element method by composing abstractions*, ACM Trans. Math. Softw., 43 (2016), pp. 24:1–24:27.
[14] J. W. Ruge and K. Stüben, *Algebraic multigrid*, in Multigrid methods, SIAM, 1987, pp. 73–130.
[15] U. Trottenberg, C. W. Oosterlee, and A. Schuller, *Multigrid*, Elsevier, 2000.
[16] R. Verfürth, *A multilevel algorithm for mixed problems*, SIAM journal on numerical analysis, 21 (1984), pp. 264–271.

II. Software & High Performance Computing

Articles in this section discuss the implementation of high performance computing (HPC) and productivity software. In many cases, performance improvements and portability are demonstrated for many-core architectures, such as conventional multicore CPUs, the Intel Many Integrated Core coprocessor (MIC), and graphical processing units (GPU).

1. *Bielich, Langou, Yamazaki, Loe, and Boman* present a one-column-at-a-time **implementation of the Householder algorithm within Trilinos** and provide performance results in the QR factorization setting.
2. *Bogle, Boman, Devine, Rajamanickam, and Slota* present several **MPI+GPU approaches for graph coloring**; essential for parallelizing scientific computations that run in distributed and multi-GPU environments. They tackle distance-2 coloring and give the first known distributed and multi-GPU algorithm for this problem.
3. *Carlson, Watkins, and Tezaur* identified performance bottlenecks related to the evaluation of the **boundary conditions in Sandia's Albany Land-Ice (ALI) code base**. They reworked how boundary conditions are represented in Albany such that all memory accesses on the GPU are now properly coalesced, resulting in performant GPU kernels.
4. *Cobb, Phipps, and Kolla* detail several implementations of optimized, performance portable **Tensor Times Matrix (TTM) kernels** utilizing the Kokkos programming model. They show how their optimized TTM kernel can competitively speed up the Sequentially Truncated Higher-Order Singular Value Decomposition (ST-HOSVD) algorithm among other tensor algorithms.
5. *Ford, Martin, Gieseler, and Cabrera-Palmer* present updates on the Instrument Characterization Catalog (CharCat), intended for the storage and visualization of characterization data pertaining to radiation detectors.
6. *Madrid Larranaga and Witzel* use **quantum circuits** as an intuitive visual representation of quantum programs and extend Prove-It (a Python-based, Sandia-developed interactive theorem prover) with quantum circuit proof capabilities.
7. *Miller, Hughes, and Cook* evaluate **DPC++**, an embedded domain specific language (eDSL), using DOE proxy applications to identify programmability gaps and performance on Intel FPGAs. They completed initial testing with the MiniAMR application from the Mantevo suite, focusing on the 7-point stencil.

A.A. Rushdi
M.L. Parks

November 1, 2020

HOUSEHOLDER ORTHOGONALIZATION IN TRILINOS

DANIEL BIELICH*, JULIEN LANGOU†, ICHITARO YAMAZAKI‡, JENNIFER LOE§, AND ERIK BOMAN¶

Abstract. Orthogonalizing a set of basis vectors is an important part of nonsymmetric iterative solvers for the solution of linear systems or eigenvalue problems. The use of Householder reflection to numerically compute an orthogonal basis is known to be unconditionally stable. For this reason, Householder reflections are used in most (if not all) state-of-the-art dense numerical linear algebra packages (e.g. LAPACK). Within this paper, we discuss an implementation of the Householder algorithm within Trilinos and provide performance results in the QR factorization setting. The framework we work is "one-column-at-a-time" or "left-looking algorithm with a block size of 1". This is a typical framework for iterative methods. The Householder orthogonalization algorithms "one-column-at-a-time" presented in this paper (Level 1 and Level 2) were explained for the first time in [6].

1. Introduction. The Householder algorithm is known to be unconditionally stable when computing an orthogonal basis numerically. As such, it is important to include this algorithm as an option for a numerical linear algebra package such as Trilinos. This report explains the work done over a Summer internship at Sandia where the main contribution was to implement the Householder orthogonalization scheme within Trilinos. Many algorithms and factorization methods can utilize Householder orthogonalization (e.g. Arnoldi expansion). In this paper, we use the Householder orthogonalization to compute the QR factorization of a tall-skinny dense matrix. This is a typical scenario for an Arnold expansion.

We have implemented two different variants of the Householder algorithm. The first variant is based on Level 1 BLAS and the second variant is based on Level 2 BLAS. It is important to note that, since we are in the "one-column-at-a-time" framework, it is not possible to derive a Level 3 BLAS variant. Both variants are implemented in C and in the Trilinos framework. We essentially use the two Householder variants (Level 1 and Level 2) as explained by Walker in 1988 in [6].

The Level 1 BLAS variant when used in the QR factorization framework is very similar to `PDGEQR2+PDORG2R` in ScaLAPACK when using 1-D distribution for tall-and-skinny matrices. Therefore we will also compare our performance results with ScaLAPACK.

Though the Level 2 variant requires more floating-point operations than the Level 1 variant, it requires much fewer synchronizations. As a result, our performance results using on up to 4096 Intel SandyBrigde CPUs demonstrate that the Level 2 variant (coded in C) scales better than the Level 1 variant or ScaLAPACK. We also compare the parallel scalability of our implementation to the classical Gram-Schmidt algorithm with reorthogonalization (CGS2), our performance results demonstrate that the Level 2 variant scales similar to CGS2 with a constant overhead. CGS2 has some fewer synchronizations and fewer FLOPS than the Level-2 variant. We have observed that, in some pathological cases in Arnoldi expansions, CGS2 suffers from numerical instability that the Householder variants do not.

The rest of the paper is organized as follows: In Section 2, we discuss the varying orthogonalization schemes used throughout this paper. We are very much interested in parallel computation so we discuss the demands to achieve this for each algorithm (synchronizations and volume of messages sent) as well as the associated complexity. We discuss the numer-

*Univeristy of Colorado Denver, daniel.bielich@ucdenver.edu
†Univeristy of Colorado Denver, julien.langou@ucdenver.edu
‡Sandia National Laboratories, iyamaza@sandia.gov
§Sandia National Laboratories, jloe@sandia.gov
¶Sandia National Laboratories, egboman@sandia.gov

ical stability for each method. The cost per iteration and as a whole for each algorithm. In Section 4, we provide performance results. We provide experiments that show the scalability and robustness of each algorithm. Provide some insight into the results presented. Within the Conclusion we summarize the paper but also give insight into the future work we intend on continuing. For details on implementation and to see the struggles we faced while implementing HH lvl2 into Trilinos, please refer to the appendix (Section A).

2. Orthogonalization Schemes. There are three main types of orthognalization schemes used for numerical computation, they fall under either (1) the Gram-Schmidt umbrella or (2) the Householder umbrella or (3) the Givens umbrella. Givens rotations are used to eliminate specific entries in a matrix, so, while we could use them in this framework, they are unlikely to perform well. When working on zero-ing out whole vectors, a Householder-based method or a Gram-Schmidt-based method is preferred. Therefore, within this document we compare two different orthogonalization schemes which are of Householder type (Algorithms 1 and 2) and one Gram-Schmidt algorithm (Algorithm 3). The Householder orthogonalization method (Level 1 or Level 2) is commonly used in LAPACK and in ScaLAPACK. CGS2 is commonly used in Trilinos. The Householder Level 1 (HH lvl1) and Householder Level 2 (HH lvl2) algorithms are used for their unconditional stability. The Classical Gram Schmidt with reorthogonalization (CGS2) is used for the ease of implementation as well as the added stability by reorthogonalizing.

2.1. Algorithms. In the following four subsections we discuss the main components needed for each different algorithm, the overall steps each algorithm undergoes for a given iteration, and the cost for each algorithm in terms of flops and communication between processes.

2.1.1. Classical Gram-Schmidt Re-Orthogonalized. The CGS2 algorithm (Algorithm 3) is the easiest scheme to implement. It requires three main steps per iteration: (1) & (2) two projections, followed by (3) a normalization. Within the algorithm, at step j, you enter the CGS2 orthogonalization function with the j^{th} column to be orthogonalized and the $j-1$ orthogonal columns ($\mathbf{Q}_{1:m,1:j-1}$) constructed in the prior $j-1$-iterations.

The projection step is to apply $(I - \mathbf{Q}_{1:m,1:j-1}\mathbf{Q}_{1:m,1:j-1}^T)$ to the column ready for orthogonalization. This is done twice. Mathematically projecting twice does absolutely nothing if the vector $\mathbf{Q}_{1:m,1:j-1}$ forms an orthonormal basis. However, in finite precision arithmetic, the first projection may lose its accuracy and therefore a second projection would then be needed.

CGS2 requires only three synchronizations per iteration, regardless of the iterate. Once for each projection (i.e the operations of the kind $\mathbf{Q}^T\mathbf{a}$). The third occurs when normalizing. So, the overall demand in communication in terms of number of reduction operations, at step j, is 3 and the aggregated volume of communication is $2j-1$.

Algorithm 3 in the Appendix shows the pseudocode of the CGS2 algorithm. Figure B.4 provides Matlab code for the algorithm with a main driver, Figure B.1.

2.1.2. Householder Level 1. The Householder Level 1 algorithm (Algorithm 2) is complex and arguably more difficult to implement than CGS2. Also it requires additional steps and additional storage compared to CGS2. This algorithm has five main components (two more than CGS2). Householder Level 1 needs $\mathbf{Q}$, $\mathbf{V}$, R and τ. $\mathbf{Q}$ is the orthogonal factor and is dense, $\mathbf{V}$ are the Householder reflections (note $\mathbf{V}$ is unit-lower triangular), R is the upper-triangular factor which represents $\mathbf{A}$ through $\mathbf{Q}$ and τ is a $1 \times j$ vector, j being the size of the current iteration. Note that, in sequential computation, one can save on storage by using the top of $\mathbf{V}$ to store R. One can take advantage of the unit triangular parameter for the BLAS function `DTRMV`. In general, in parallel computation, in iterative

methods, the R-factor is replicated on all processes. We follow this scheme, so we do not store "R on top of V" (even on process 0).

The algorithm contains three overall steps per iteration. Consider step j, at this step we have constructed $\mathbf{Q}_{1:m,1:j-1}$, $\mathbf{V}_{1:m,1:j-1}$, $R_{1:j-1,1:j-1}$, $\tau_{1:j-1}$. The goal is to append the j^{th} column to $\mathbf{Q}, \mathbf{V}$ and R, as well as compute τ_j. The way to accomplish this is, first apply the previous $j-1$ Householder reflections to update $\mathbf{a}_j$, this is done sequentially (i.e. $\prod_{i=1}^{j-1}(I - \mathbf{V}_{1:m,i}\tau_i\mathbf{V}_{1:m,i}^T)$). After this operation $\mathbf{a}_j$ is updated. The top $j-1$ elements are exactly $R_{1:j-1,j}$, meaning the appended column to R is almost finished, the j^{th} element, $r_{j,j}$, will be constructed during the second step. The following step is to construct τ_j and the j^{th} Householder reflection. Please refer to Algorithm 2 to see the details. The final step is to construct and append the orthogonal column $\mathbf{Q}_{1:m,j}$. Which is applying the now j Householder reflections in reverse order, applied to the the j^{th} basis vector e_j.

This algorithm requires $2j$ synchronizations per iteration, $j-1$ from the first step, updating $\mathbf{a}_j$, two synchs from the second step (a dot product of the updated column $\mathbf{a}_j$ and broadcasting $V_{j,j}$ so we can construct the correct τ_j and the corresponding Householder reflection). Finally there are $j-1$ synchs for constructing the orthogonal factor. We get away with not adding a synch when applying $(I - \mathbf{V}_{1:m,j}\tau_j\mathbf{V}_{1:m,j}^T)\mathbf{e}_j$ because $\mathbf{V}_{1:m,j}^T e_j = 1$. So we get away with setting $\mathbf{Q}_{1:m,j} = e_j - \tau_j\mathbf{V}_{1:m,j}$ avoiding a communication. Note the aggregated volume of communication sent is $2j$, each inner-product produces 1 element that needs to be reduced. There is an additional cost of size j to send the new column of R to each process. Thus the aggregated volume of communication for HH lvl1 is $3j$.

Algorithm 2 in the Appendix shows the pseudocode of the level-1 algorithm. Figure B.3 provides Matlab code for the algorithm with a main driver, Figure B.1.

2.1.3. ScaLAPACK. For comparison, we use the ScaLAPACK subroutine `PDGEQR2+PDORG2R`. This should perform the same operations as HH lvl1. The main difference is that the ScaLAPACK interface is asking for the whole matrix $\mathbf{A}$ at the start. (As opposed to access $\mathbf{A}$ and return $\mathbf{Q}$ "one-column-at-a-time".) This is a major difference from an interface point of view, and, therefore ScaLAPACK `PDGEQR2+PDORG2R` cannot be used in the context of iterative methods like, e.g., GMRES. That being said, the algorithms within ScaLAPACK `PDGEQR2+PDORG2R` are working "one-column-at-a-time" and should therefore behave very closely to HH lvl1.

For disclaimer, it is possible to use ScaLAPACK to obtain a real "one-column-at-a-time" interface. The sequence of operation would be something like `PDORM2R` to push in the small world (applying the Householder reflections to update, this zero's out the first $j-1$ elements in the updated column and is thus a small world), `PDLARFG` to create the r column and the $\mathbf{v}$ vector, `PDORG2R` to construct $\mathbf{q}$, and `PDORM2R` to pop in the big world (we get the top $j-1$ elements, putting us back into a big world). We are mainly interested in performance comparison. So, we thought comparing to ScaLAPACK `PDGEQR2+PDORG2R` would be good enough.

ScaLAPACK `PDORG2R` and `PDORG2R` are implementations of right-looking algorithms. Using a right-looking algorithm is possible only when all of the columns of $\mathbf{A}$ are given to the routine `PDGEQR2` when called. It is important to understand that in the "one-column-at-a-time" paradigm, we do not have all the columns at once, so we cannot use a right-looking algorithm. We do nevertheless compare to ScaLAPACK as a reference point for performance. When ScaLAPACK applies a Householder matrix, the algorithm applies the matrix to the trailing columns of the input matrix $\mathbf{A}$. Additionally, when constructing $\mathbf{Q}$ using `PDORG2R`, ScaLAPACK takes advantage of having all Householder reflections. First, the routine applies the last constructed Householder matrix $H(n)$ (note: $H(n) = (I - \mathbf{v}_n\tau_n\mathbf{v}_n^T)$) to e_n (the n^{th} standard-basis vector). The next step the algorithm applies $H(n-1)$ to $[e_{n-1};\ H(n)\cdot e_n]$.

Through this ScaLAPACK can take advantage of structure by reversing the order of how the columns of the orthogonal factor is constructed. So ScaLAPACK constructs q_n first, then q_{n-1}, etc. The order for ScaLAPACK is v_1, v_2, ...,v_n, q_n, q_{n-1},...q_1. This is not possible in our "one-column-at-a-time" framework in the sense that once v_1 is constructed we must construct q_1 The order for HH lvl1 and lvl 2 is v_1, q_1, v_2, q_2, v_3, q_3, ...,v_n, q_n.

2.1.4. Householder Level 2. Householder Level 2 consists of five main steps when dealing with parallel computation.

1. Apply the previous Householder reflections to update the current column;
2. Broadcast R (the triangular factor in a QR decomposition - only needed for parallel computation);
3. Construct the new Householder reflection;
4. Construct the next column in the orthonormal basis;
5. Construct and broadcast the next column in the triangular factor T, a consequence of blocking Householder reflections.

At a given iteration, with these five steps, the algorithm constructs the new Householder reflection vector, the new orthonormal basis vector and the next column within the upper-triangular factors R and T.

This algorithm differs from HH lvl1 by instead of applying the Householder reflections sequentially, both when updating $\mathbf{a}_j$ and constructing $\mathbf{Q}_{1:m,j}$, these operations are blocked. Please refer to the Appendix, section A for a better understanding. A consequence of blocking these Householder reflections is, there is an upper-triangular factor T needed to keep the algorithm mathematically equivalent, i.e. $(I-\mathbf{v}_1\tau_1\mathbf{v}_1^T)\cdots(I-\mathbf{v}_n\tau_n\mathbf{v}_n^T) = (I-\mathbf{V}T\mathbf{V}^T)$. Let's evaluate projecting two Householder matrices,

$$\begin{aligned}
&\left(I-\mathbf{v}_1\tau_1\mathbf{v}_1^T\right)\left(I-\mathbf{v}_2\tau_2\mathbf{v}_2^T\right)\\
&= I-\mathbf{v}_1\tau_1\mathbf{v}_1^T-\mathbf{v}_2\tau_2\mathbf{v}_2^T+\mathbf{v}_1\left(\tau_1\mathbf{v}_1^T\mathbf{v}_2\tau_2\right)\mathbf{v}_2^T\\
&= I-[\mathbf{v}_1;\ \mathbf{v}_2]\begin{bmatrix}\tau_1 & \tau_1\mathbf{v}_1^T\mathbf{v}_2\tau_2\\ 0 & \tau_2\end{bmatrix}[\mathbf{v}_1;\ \mathbf{v}_2]^T
\end{aligned}$$

which shows the additional term needed to keep the algorithm equivalent while blocking. Now let us consider an arbitrary step j where we have $j-1$ columns in $\mathbf{V}, T$.

$$\begin{aligned}
&\left(I-\mathbf{V}_{1:m,1:j-1}T_{1:j-1,1:j-1}\mathbf{V}_{1:m,1:j-1}^T\right)\left(I-\mathbf{v}_j\tau_j\mathbf{v}_j^T\right)\\
&= I-\mathbf{V}_{1:m,1:j-1}T_{1:j-1,1:j-1}\mathbf{V}_{1:m,1:j-1}^T-\mathbf{v}_j\tau_j\mathbf{v}_j^T\\
&\quad+\mathbf{V}_{1:m,1:j-1}\left(\tau_jT_{1:j-1,1:j-1}\mathbf{V}_{1:m,1:j-1}^T\mathbf{v}_j\tau_j\right)\mathbf{v}_j^T\\
&= I-[\mathbf{V}_{1:m,1:j-1};\ \mathbf{v}_j]\begin{bmatrix}T_{1:j-1,1:j-1} & T_{1:j-1,1:j-1}\mathbf{V}_{1:m,1:j-1}^T\mathbf{v}_j\tau_j\\ 0 & \tau_j\end{bmatrix}[\mathbf{V}_{1:m,1:j-1};\ \mathbf{v}_j]^T.
\end{aligned}$$

This shows exactly the formula to construct a column in T

$$T_{1:j-1,j} = T_{1:j-1,1:j-1}\mathbf{V}_{1:m,1:j-1}^T\mathbf{V}_{1:m,j}\tau_j \tag{2.1}$$

$T_{j,j} = \tau_j$, the diagonal of T is exactly the elements in the $1\times j$ vector τ used in the Householder Level 1 algorithm.

The benefit of blocking these operations is that we can reduce the cost of communication. The drawback is an increased cost in floating point operations, due to the need to construct T, and to apply T; as well as a cost in storage to store T because each process needs it. Householder Level 2 has more FLOPs than Householder Level 1 but has a constant cost in communication (constant means independent of j). The algorithm has five

synchronizations that correspond to the five steps mentioned above. At step j, one synch happens when updating $\mathbf{a}_j$. The second synch is when we construct τ_j and $\mathbf{V}_{1:m,j}$. The third happens when constructing $\mathbf{Q}_{1:m,j}$, the fourth when constructing $T_{1:j-1,j}$ and finally the fifth synch happens when broadcasting $R_{1:j,j}$ to each process. Hence the aggregated volume of communication is $4j$ at iteration j.

Algorithm 1 in the Appendix shows the pseudocode of the level-2 algorithm. Figure B.2 provides Matlab code for the algorithm with a main driver, Figure B.1.

2.2. Numerical stability. Below we provide a table (Table 2.1) which shows what we can expect, as far as numerical stability, from each algorithm presented. Please refer to [4], [6] and page 361 in [5]. In the context of the QR Factorization (the column regarding representativity) we would expect each method to satisfy $\|\mathbf{A}-\hat{\mathbf{Q}}\hat{R}\|_F/\|\mathbf{A}\|_F \leq c_1(m,n)\mathcal{O}(\mu)$ Where c_1 is some function of m and n. μ is the precision of the hardware setup in use (we use double-precision so expect $\mu \approx 10^{-16}$) and $\|\cdot\|F$ stands for the Frobenius norm. The hat on $\mathbf{Q}, R$ implies they are computed quantities. The column for orthogonalization shows what one can expect for the error produced in evaluating the orthonormality of $\mathbf{Q}$. We evaluate the following after each run, $\|I - \hat{\mathbf{Q}}^T\hat{\mathbf{Q}}\| \leq c_2(m,n)\mathcal{O}(\mu)$. Where c_2 is some function of m and n.

We note that the bound on orthogonality for CGS2 has only be proven true under the assumption that A is numerically nonsingular. However it seems that CGS2 is numerically stable in practice for many matrices (including many of the ones that are numerically nonsingular). It seems that, counter-intuitively, numerical errors are actually helping CGS2 being more stable. (Trying to quantify this in a probabilistic sense would actually be a nice research project.) We have observed that, in some pathological cases coming from Arnoldi expansions, CGS2 suffers from numerical instability that the Householder variants do not. So the bound in Table 2.1 for orthogonality of CSG2 is false. But, we believe that, in a probabilistic sense, they are correct. And we have observed that they are more representative of what one is likely to observe. And they have been proven correct if one assumes that A is numerically nonsingular.

Orthogonalization Scheme	Representativity	Orthogonalization
CGS2	$\mathcal{O}(\mu)$	$\mathcal{O}(\mu)$
HH Level 1	$\mathcal{O}(\mu)$	$\mathcal{O}(\mu)$
HH Level 2	$\mathcal{O}(\mu)$	$\mathcal{O}(\mu)$

Table 2.1: Level of Orthogonalization and Representativity one can expect from each method.

2.3. Communication and Computation Costs. The following tables (Table 2.2 and Table 2.3) show, for each algorithm, the highest demanding terms of cost in communication, number of All-Reduce type operations (synchs) and the size of the messages sent per iteration (volume). HH lvl1 and CGS2 both share the same number of FLOPs, but the number of synchronizations vary greatly. Especially when j is large. HH lvl 2 on the other hand has more demand in two of the three categories. The number of synchronizations per iteration remains constant though, which is nice for any communication bound problem. The reason HH lvl 2 has a greater demand in FLOPs and volume is the need to construct T, so blocking the Householder reflections is possible. Please refer to section 5 for details regarding how we can achieve T for free essentially, providing one wants to construct the orthogonal factor $\mathbf{Q}$ and not only R.

Orthogonalization Scheme	FLOPs per Step	Synchs	Volume
CGS2	$8(m/p)j$	3	$2j$
HH Level 1 left-looking	$8(m/p)j$	$2j$	$3j$
HH Level 1 right-looking	$8(m/p)(n-j)$	$2(n+j)$	$3(n+j)$
HH Level 2	$10(m/p)j + 3j^2$	5	$4j$

Table 2.2: Highest demanding term for FLOPs, Synchs and the Volume at iteration j.

Orthogonalization Scheme	FLOPs	Synchs	Volume
CGS2	$4(m/p)n^2$	$3n$	n^2
HH Level 1 left-looking	$4(m/p)n^2$	n^2	$\frac{3}{2}n^2$
HH Level 1 right-looking	$4(m/p)n^2$	n^2	$\frac{3}{2}n^2$
HH Level 2	$5(m/p)n^2 + n^3$	$5n$	$2n^2$

Table 2.3: Highest demanding term for FLOPs, Synchs and the Volume after a full factorization.

2.4. Memory footprint. At step j, CGS2 takes as input one column of $\mathbf{A}$ and $j-1$ columns of $\mathbf{Q}$ and outputs the j-th column of $\mathbf{Q}$. So CGS2 needs mj memory space to operate at step j.

At step j, HH takes as input one column of $\mathbf{A}$ and $j-1$ columns of $\mathbf{V}$ and output the j-th column of $\mathbf{V}$, and the j-th column of $\mathbf{Q}$. From then, there are three options: (1) one can decide to keep all the vectors of $\mathbf{Q}$ generated, in which case storage would be $2mj$, (2) one can decide to reconstruct the vectors $\mathbf{Q}$ in place at a later time from the vectors $\mathbf{V}$, this is certainly possible but is somewhat extreme, the storage would only be $\mathbf{Q}$, this is not really the modus operandi that we have in mind, so we skipped this option, (3) one can use the constructed vector $\mathbf{q}$ for the next iteration (in Arnoldi expansion for example) and then keep on reusing this memory space as iteration comes along. The storage would be $m(j+1)$. So close to CGS2. This is the modus operandi we have in mind. Please note that we can construct linear combination of the vectors $\mathbf{Q}$ pretty easily and cheaply from $\mathbf{V}$ without reconstructing the full $\mathbf{Q}$.

3. Experiment Setups. We have implemented the Classical Gram Schmidt and Householder orthogonalization algorithms in C using BLAS and MPI. We also have another implementation of the algorithms based on Trilinos software package framework. In particular, for matrix and vector operations on the distributed-memory computer, we used Tpetra software package [1] and MultiVector traits interfaces of Belos software package [2]. Both implementations are publicly available, and more details of our implementation are provided in Appendix A. We first compare the performance of our C implementation with Householder implementation (`PDGEQR2` and `PDORG2R`) of ScaLAPACK. We then compare the performance of our C and Trilinos implementations.

We conducted our numerical experiments on SkyBridge cluster at Sandia National Laboratories. Each compute node of SkyBridge has 16-core 2.6 GHz Intel SandyBridge CPU and 64 GB of main memory. These compute nodes are connected through QDR Infiniband. We compiled our code using Intel 2020 2.254 compiler with the optimization flag `-O3`, and linked to Intel Math Kernel Library (MKL) and OpenMPI-Intel 4.0.3.

For our performance studies, we focus on the tall-skinny matrices ($m \gg n$), and hence,

we used the random matrices with the following three sets of sizes:

1. $m = 3,000,000$ and, $n = 50$;
2. $m = 20,000,000$ and, $n = 50$;
3. $m = 50,000,000$ and, $n = 50$.

We have verified the correctness of our implementations by checking the resulting representation and orthogonality errors (i.e., $\|\mathbf{A} - \mathbf{Q}R\|_F / \|\mathbf{A}\|_F$ and $\|\mathbf{Q}^T\mathbf{Q} - I\|_F$) were of the order of machine precision.

We report the fastest time from five runs. There are two experiments repeated three times in each of the discussions. The first experiment is the left column of plots, which is a performance experiment. The second experiment is the right column of plots, which consist of the time to solution for performing the QR factorization on a dense tall-skinny matrix. These two experiments we repeated three times, the number of nodes used remains constant, 1 node to 256 nodes (doubling for each data point). The size of the matrix varies for the different runs. The first row corresponds to a matrix size of $m = 3 * 10^6$, the second row $m = 20 * 10^6$ and the third row $m = 50 * 10^6$.

4. Performance Results. We study the strong scalability of the three orthogonalization algorithms; Classical Gram-Schmidt with Reorthogonalization (CGS2) in Algorithm 3, and Householder level 1 (HH lvl1) and level 2 (HH lvl2) in Algorithms 2 and 1, respectively.

The Level 2 HH code is called "5-synch" in the performance plots, and the Level 1 HH code is called "many-synch".

We first compare the performance of our C implementations of different orthogonalization schemes in Section 4.1. We then compare the performance with our Trilinos implementations and with the Householder implementation in ScaLAPACK, specifically the routines for a panel QR factorization, i.e., the QR factorization of a tall-skinny matrix (`PDGEQR2` and `PDORG2R`).

4.1. Strong Parallel Scaling Studies with C code. Here, we evaluate the C implementation of three orthogonalization algorithms, HH lvl1, HH lvl2 and CGS2. Refer to Section A for details of the implementation of the code.

Figures 5.1 show the strong scalability of the C implementations, using three different numbers of rows, i.e., $m = 3,000,000$, $m = 20,000,000$, and $m = 50,000,000$, respectively, while the number of columns is fixed at $n = 50$. For each figure, the left plots shows the performance (GFLOP/s) per node, while the right plots show the total time to compute the QR factorization of a panel. The x-axes remains constant through all of these plots. Which vary by number of nodes, starting with one node, moving to 256 nodes for the same problem size. Each node on SkyBridge has 16 cores available. So in terms of cores we evaluate using up to as many as 4096 cores. The y-axis for the performance plots represents GFLOPs/second/node. To compare each method equally we set GFLOPs to $4mn^2$. The y-axis for the time to solution plots is the total time for each run. Each method has an associated dashed line that represents where the perfect scalability would be (given the initial run on one node). The reason we provide both experiments side by side is, when we start to lose performance per node, we have no idea how bad it can really get. Looking at the timing results, we more clearly see how bad things can become as far as scalability. We find it useful to have both plots side-by-side depending on the regime we are looking at.

We see that HH level 2 (cyan, "5-synch") is parallel to CGS2, which is consistent in each experiment. This is good and this is what we were expecting to observe. HH level 2 has a 20% FLOPs overhead with respect to CGS2 and so we clearly see this 20% overhead. Otherwise the curves are somewhat similar and parallel. Although HH lvl1 perform a similar number of FLOPs as CGS2, its performance relies on Level 1 BLAS kernels and many synchronization, so the performance curve are quite different.

We see a very strongly marked cache effect for all methods. Which leads to strong linear speedup. We explain with HH lvl1 but this is true for all three methods. (And will also be true for Trilinos and ScaLAPACK code.) Let us look at HH lvl1, the brown line, in the cases $m = 3,000,000$, $m = 20,000,000$, and $m = 50,000,000$. For each case, on one node, HH lvl1 starts between 5-10 GFLOPs per second per node. Of the three experiments there is a similar performance boost. For plots 5.1 (the top two) the peak happens when moving from one to two nodes. For plots 5.1 (the middle two plots) this begins starting after a run on four nodes and for plots 5.1 (the bottom two) the performance boost occurs after a run on eight nodes. Each of the peaks land approximately 20 GFLOPs per second per node. The reason for this result is most likely a cache effect. Looking at the peaks, for $m = 3 * 10^6$, with 2 nodes, each core gets 93,750 elements from each column and a matrix with $n = 50$ columns with double precision number requires 572 MB of memory per core, for $m = 20 * 10^6$, with 16 nodes, each core gets 78,125 elements and a matrix requires 572 MB of memory per core, and, for $m = 50 * 10^6$, with 64 nodes, each core gets approximately 48,828 elements and a matrix requires 298 MB of memory per core. If we look at the scalability curves (the plots on the right), each cache effect yields a superlinear speed up.

For $m = 3,000,000$, the peak for Level 2 (cyan, "5-synch") and Level 1 (brown, "many-synch") are exactly at the same point which makes sense since they use the same number of matrices. We have a harder time to interpret the peak across methods. CGS2 uses one matrix. Indeed, the matrix of $\mathbf{Q}$ overwrites the matrix of $\mathbf{A}$. This is somewhat akin with Householder, where we have one input vector a, an input matrix $\mathbf{V}$ and, in output, $\mathbf{V}$ gets augmented by one column and input $\mathbf{a}$ is transformed in $\mathbf{q}$. So the memory footprint of CGS2 and HH should be somewhat similar. Yet the peak (cache effect) happened at quite different places. We do not have a good explanation for this. It is likely that our implementation of HH needs to fit two matrices in cache ($\mathbf{A}/\mathbf{V}$ and $\mathbf{Q}$) to benefit from cache effect, while CGS2 only needs to fit one matrix in cache ($\mathbf{A}/\mathbf{Q}$)) to benefit from cache effect. Related to memory footprint, we also note that all codes have a third matrix that holds a copy of $\mathbf{A}$ for checking purpose. We do not think this should impact cache effect, but mention this in passing.

In the top plots of Figure 5.1 we see all three algorithms drop in performance as we distribute across 256 nodes. When we evaluate the left plot to the corresponding right plot, we see the loss in performance relates to the loss of scalability. When each method starts to dip down in the performance plot, the trend in the time to solution increases showing the problem becomes communication bound. For the second experiment (middle two plots), we notice HH lvl2 and CGS2 scale perfectly but HH lvl1 does not. The two methods that scale have a constant number of synchronizations per iteration where HH lvl1 is iteration dependent.

Looking at the third experiment, we see that each method does not scale perfectly as we increase to 256 nodes and that HH lvl1 and HH lvl2 follow similar paths. The reason that each method has a similar time to solution is because the problem becomes FLOP dependent. CGS2 and HH lvl1 both share approximately $4mn^2$ FLOPs and HH lvl2 about $5mn^2$. The extra demand of communication for HH lvl1 is why it follows more HH lvl2 versus CGS2. The dip for CGS2 on plot bottom left plot for 16 nodes is strange. This could be noise and rerunning the experiment could remove the dip all together.

4.2. Strong Parallel Scaling Studies with Trilinos. When evaluating the performance results in Trilinos (and ScaLAPACK) we see similar behavior. The same cache effect that happened for HH lvl1 implemented in C occurs for the implementation within Trilinos. There is a boost in performance and then when the problem becomes communication bound, the scalability and performance is lost. HH lvl1 also shares the same trend in Trilinos as

in C for the time to solution plots. A U-shape curve where we see super-linear speed up caused by the cache effect, followed by an increase in timings because the method does not scale.

We note that the performance of ScaLAPACK somewhat follows the Trilinos implementation for CGS2 and HH lvl2, and does not follow HH lvl1. ScaLAPACK does not use the T-matrix. So how can ScaLAPACK have Level 2 performance without a T matrix? The reason is that ScaLAPACK breaks our paradigm of "one-column-at-a-time". ScaLAPACK requires all n columns at once. And the algorithm of ScaLAPACK cannot be used in the "one-column-at-a-time" framework. Since ScaLAPACK has all n columns at once, ScaLAPACK can implement a right-looking Level 2 algorithm without a T matrix. This is why the performance of ScaLAPACK is closer to Level 2 than Level 1. In the "one-column-at-a-time" framework, we must use a T matrix to enable Level 2 performance.

One thing to note - the C implementation scales for the methods that had constant number of synchronizations per iteration, except for the experiment when $m = 3 * 10^6$. The $m = 3 * 10^6$ problem is very small, and with a strong scaling experiment, we are really reaching the limit of what can be done. For $4,096$ cores, each core has $m_{\text{loc}} = 733$ entries per vector and this is not much. We clearly see the effect of communication in this case. The C implementation is lightweight, only calling MPI_Allreduce when communication is needed, so scales relatively well across the board. We observe that HH lvl2 and CGS2 implemented in Trilinos as well as ScaLAPACK do not scale when using many processes. We think that there could be an overhead when calling Trilinos and ScaLAPACK. But we could also be doing something that is considered a bad usage within Trilinos, we are still trying to fully understand this behavior. ScaLAPACK follows the same trend as Trilinos.

Also, note that for Trilinos HH lvl2 and HH lvl1 there is no data point at one node for the experiment when $m = 50 * 10^6$. This is because the algorithm requires an additional mj in storage for the Multivector V as opposed to the Trilinos implementation of CGS2. This additional storage demand resulted in there not being enough memory to distribute the matrix across 16 cores.

5. Conclusions and Future Work. In this paper, we have studied the performance of three different orthogonalization algorithms, two of which are based on Householder orthogonalization. Our preliminary results indicate that each method performs more or less as we would expect. As expected, HH lvl1 does not scale on any of the experiments presented where the C implementation of HH lvl2 and CGS2 does indeed scale. There appears to be some overhead using Trilinos and ScaLAPACK. We observe super-linear speed up at times. The main difference in terms of performance between HH lvl2 and CGS2 seems to be the fact that HH lvel2 performs 20% more FLOPS than CGS2.

We are currently working on reducing the number of FLOPS for HH Level 2 and reducing the number of synchronizations. The computation required to construct a column in T is used in the construction of the same column in $\mathbf{Q}$. We can reuse this computation and save $2(m/p)j$ FLOPs at each iteration, as well as reduce one communication. This would put the highest demanding term for FLOPs of HH lvl2 to be the same as CGS2. But we do not want to stop there. Within C we have a version of HH lvl2 that requires only two synchronizations per iteration. It also has the reduced FLOPs implemented. There is some struggle on the Trilinos side. In C, we are able to manipulate values before an All_Reduce occurs. Trilinos, on the other hand, takes care of all MPI processes for us. So we cannot manipulate values prior. There still might be a way to achieve two synchronizations per iteration using Householder within Trilinos, this is the direction we are moving now.

Our plan for future work is to implement the low-synch reduced-FLOPs Householder algorithim within Trilinos for use by solvers. We were able to create a reduced FLOPs

version (before we ran into the issue of not scaling - refer to Appendix A for struggles in constructing the algorithm within Trilinos) but the reduction of FLOPs by reusing the computation required an added broadcast so we did not reduce a communication. We are working on finding a way to broadcast these elements when we broadcast the column in R after updating. We also have a 1-synch CGS2 algorithm in MPI+C that we want to implement and study in the framework of Trilinos. Also, we want to study all these algorithm in the GPU setting. Indeed we have not yet noticed much improvement by reducing the number of synchronizations. We believe that, by exploiting GPUs, we will better illustrate why reducing the number of synchronization of an algorithm is important.

Finally, all this work has been focused on "one-column-at-a-time", but we really want to extend the work to "many-columns-at-a-time". This should not be too hard, this should be very satisfying and provide good speedup, we believe that the use of Householder (as opposed to Gram-Schmidt) in the "many-columns-at-a-time" is even more justified and we are looking forward to be able to move in this direction. The "many-columns-at-a-time" has direct application in the context of block iterative methods, in particular for eigendecomposition or singular value decomposition, and in the context of s-step methods [3].

REFERENCES

[1] C. Baker and M. Heroux, *Tpetra, and the use of generic programming in scientific computing*, Scientific Programming, 20 (2012), pp. 115–128.
[2] E. Bavier, M. Hoemmen, S. Rajamanickam, and H. Thornquist, *Amesos2 and Belos: Direct and iterative solvers for large sparse linear systems*, Scientific Programming, 31 (2012), pp. 241–255.
[3] A. Chronopoulos and C. Gear, *s-step iterative methods for symmetric linear systems*, Journal of Computational and Applied Mathematics, 25 (1989), pp. 153 – 168.
[4] L. Giraud, J. Langou, and M. Rozložník, *The loss of orthogonality in the Gram-Schmidt orthogonalization process*, Computers and Mathematics with Applications, 50 (2005), pp. 1069–1075.
[5] N. Higham, *Accuracy and Stability of Numerical Algorithms*, vol. 94, 01 2002.
[6] H. Walker, *Implementation of the GMRES method using Householder transformations*, Siam Journal on Scientific and Statistical Computing, 9 (1988).

Appendix A. Implementation.

We have various versions which we compare against. The orthogonalization schemes used within this paper are the two Householder algorithms (Level 1 and Level 2) and the Classical Gram Schmidt algorithm with Reorthognalization. For HH lvl1, HH lvl2 and CGS2 we have one algorithm built in the framework of Trilinos, one built in C using BLAS operations and MPI. In addition we compare to ScaLAPACK's implementation of Householder using `PDGEQR2` and `PDORG2R`.

A.1. Implementing Householder within C. The goal of the C implementation is to have lightweight, easy to read and understand, portable code. We believe we succeeed in these goals. We started the C implementation first, and then moved on to a Trilinos implementation. The C implementation is a good point of reference for timing comparison and scalability, it is hard to do a more lightweight code. We directly acces BLAS, LAPACK and MPI and the structure of the vectors is as simple as it can be and native to C. The issue with the C implementation is that (1) we do not have many matrix-vector products implemented for Arnoldi expansion (only one), (2) we do not have preconditioner implemented, and (3) we do not have GPU implementation, and, in general, we will be very limited with hetereogeneous computing and handling a mix of MPI processes, CPU threads and GPU computing, right now the parallelism is only MPI-based. A Trilinos implementation should solve these three limitations and enable more interesting experiments. That being said the lightweight C code is a good point of reference, also some readers might find a C code more useful than a Trilinos code to replicate our work.

For C we use BLAS operations - `DTRMV`, `DGEMV`, `DDOT`, `DAXPY`. The difference between versions of Householder, HH Level 1 uses `DAXPY` for the projection step and HH Level 2 uses `DGEMV` for these steps.

To make the code parallel, we add MPI processes during the five main steps of the algorithm. Meaning when we first update we call `DGEMV` (for HH Level 2) for the operation $\mathbf{V}_{1:m,1:j-1}^T \mathbf{A}_{1:m,j}$. Then this is followed by an MPI_ALLreduce with a volume of $j-1$. HH Level 1 does that operation $j-1$ times (once per Householder reflection) using the `DAXPY` and has an MPI_Allreduce each time with a volume of size 1.

The C implementation also takes advantage of the structure from using Householder reflections. One way we simplify this within the C implementation, we assume the factorization is never too large and that we are working on tall-skinny matrices. If that is the case, the upper triangular block of $\mathbf{V}$, will always live on process zero which allows for easy manipulation. Before each operation we have one if else statement to separate the dense operations from the one with structure. Meaning we check if we are on process zero or not. If we are on process zero, we do a `DTRMV` on the top square block and then a `DGEMV` on the lower rectangular block. If the process is not process zero, then it just does `DGEMV` using the size of its local m.

A.2. Implementing Householder within Trilinos. Implementing Householder (Level 1 and Level 2) follows a similar design as for Trilinos. The objects needed for the algorithm remain the same, the steps within the algorithm are the same. What is different is the functions we use. The algorithms made in the framework of Trilinos is built using Tpetra and Belos MultiVector traits. The main objects for the algorithms are SerialDense matrices and MultiVectors.

While working to implement the Householder algorithm (Algorithm 1) in Trilinos, obstacles occurred and lessons were learned. After constructing the algorithm using Tpetra traits, we began testing and, through some strong scalability experiments, we found that the implementation did not scale.

There is structure within the Householder algorithm that one can exploit. The implementation of Householder in C exploits this structure and it is how we initially began coding HH lvl2 in Trilinos. But to accomplish this requires additional steps in Trilinos compared to in C. Because $\mathbf{V}$ is unit-lower triangular we can in general offset `DGEMV` operations to apply only to the lower rectangular block. Then we can use `DTRMV` to achieve a triangular solve and add the two operations to avoid computing floating point operations on zeros and ones. (And to avoid storing these zeros and ones.) In Trilinos though, offset-ing on the rows of MultiVectors is more difficult than to offset on columns. Let us start from the beginning so we can understand what struggles led to this version that is not scalable.

At the beginning, we started this endeavor using Belos only functions and constructed each matrix as a SerialDense matrix. After this was set up correctly and the algorithm could do the QR factorization on an arbitrary dense matrix, we evaluated how to transition the tall and skinny matrices to MultiVectors $(\mathbf{V}, \mathbf{Q}, \mathbf{A})$. The issue with only using Belos traits, in order to do the Householder Level 1 and 2 algorithms we need to access elements in these vectors. At this point we realized we needed to transition from Belos specific to using the Tpetra MultiVector traits, where this functionality existed. From our understanding, Kokkos objects are not compatible with the Tpetra functions.

The three main functions in Belos used to project and normalize are `MvTransMv`, `MvTimesMatAddMv` and `MvDot`. We also use the BLAS operation `DTRMV` when applying the T factor. As mentioned in the algorithm section, to update the incoming column correctly we need to offset two MultiVectors (the first $j-1$ columns in $\mathbf{V}$ and the the j^{th} column of $\mathbf{A}$ - the column ready to be updated). This is easily accomplished using the

Teuchos Range1D function to get a range of columns to grab and using the Belos trait CloneViewNonConst to clone the column(s) we want. Because of this to correctly use V, we need either to be able to use `DTRMV` with a MultiVector by offsetting in row-space or we need to explicitly put zeros in the upper-triangular part of $\mathbf{V}$ and one's on the diagonal. The BLAS function `DTRMV` is not compatible with a MultiVector object so that is out of the question. Because of this we could not implement the offset in the row-space and use `DTRMV`. There is one other way to work around this though. We can broadcast the lower-triangular region and store it underneath the R factor. Then it is stored in a SerialDense matrix and still could be a useful way to do the factorization. This is what we began to work on.

We found in order to offset in the rows of a MultiVector we need to create a submap. Creating a submap at each iteration is not scalable. What we did, each process needs to know the start of their world. Meaning at what point in the row of a given MultiVector does that specific core hold, each core needs to know the start and end of their own worlds. Then we can offset to the lower rectangular block on $\mathbf{V}$ and have the top square - the lower triangular part in a SerialDense matrix. The downside is at iteration j we need to create a submap that points to the $j + 1^{\text{st}}$ element through m, for both the $j - 1$ Householder reflections and the column about to updated. The issue becomes, we also need to update the top j elements so we can correctly construct and broadcast the new column being appended to R.

Initially we did not know the creation of the submaps was not scalable. So we did this twice each iteration in the HH lvl2 algorithm. Which allowed us to offset in the rows of a MultiVector and exploit some of the structure the Householder algorithm provides. When doing the performance results we saw that the algorithm did scale so timed each component separately and this is where we found what we needed to remove. But to remove this also forced us to do operations on a full block and ignore the structure.

Now we set the diagonal in $\mathbf{V}$ to 1 and the elements above to 0. But before we change these elements we broadcast the j elements to every process. To do this in Trilinos we CloneCopy the j^{th} column and use the doImport function which allows each process to gain access and use getLocalViewHost so we can copy these elements out of a MultiVector into a SerialDense object. The change that allowed us to remove the creation of a submap at each iteration was to put it outside of the loop. We do need to know the size of the factorization for the panel, because before we begin the loop of orthogonalization we create one submap that is of size n and pulls the entier $j \times n$ block when we broadcast. We only access the j^{th} column of this broadcasted block though.

Appendix B. Matlab Implementation.

Here we provide Matlab code that does each algorithm presented, Householder Level 1, Householder Level 2 and Classical Gram-Schmidt with Reorthogonalization. We, in addition, provide a driver for each algorithm. One can copy and paste these as .m files, you only need to uncomment one and exactly only one line associated with the method you want to use in the main. The kind of interface is convenient for an orthogonalization toolbox and it is for example convenient to build an iterative solver around this interface.

Appendix C. Additional Plots Using Data from Plots Presented.

Appendix D. Pseudocodes.

Algorithm 1 Householder Orthogonalization Level Two (HH LVL2) Algorithm.

1: **for** $(j = 1; j <= n; j++)$ **do**
2: $\mathbf{v} = \mathbf{a}_j$
3: **if** $(j > 1)$ **then**
4: work $= V^T_{1:m.,1:j-1}\mathbf{v}$
5: work $= T^T_{1:j-1,1:j-1}$work
6: $\mathbf{v} = \mathbf{v} - \mathbf{V}_{1:m.,1:j-1}$work
7: **end if**
8: $\alpha = \mathbf{v}^T_{j+1:m}\mathbf{v}_{j+1:m}$
9: $\beta = \sqrt{\alpha + \mathbf{v}_j^2}$
10: $\gamma = \mathbf{v}_j + \beta * \text{sign}(\mathbf{v}_j)$
11: $R_{1:j-1,j} = \mathbf{v}_{1:j-1}$
12: $R_{j,j} = -\beta * \text{sign}(\gamma)$
13: $T_{j,j} = \tau_j = \frac{2}{1+(\alpha/\gamma^2)}$
14: $\mathbf{V}_{j+1:m,j} = \frac{1}{\gamma}\mathbf{v}_{j+1:m}$
15: $\mathbf{V}_{j,j} = 1.0$
16: $\mathbf{V}_{1:j-1,j} = 0.0$
17: $\mathbf{Q}_{j+1:m,j} = -\tau_j * \mathbf{v}_{j+1:m}$
18: $\mathbf{Q}_{j,j} = 1 - \tau_j$
19: $\mathbf{Q}_{1:j-1,j} = 0.0$
20: **if** $(j > 1)$ **then**
21: work $= \mathbf{V}^T_{1:m.,1:j-1}\mathbf{Q}_{1:m,j}$
22: work $= T_{1:j-1,1:j-1}$work
23: $\mathbf{Q}_{1:m,j} = \mathbf{Q}_{1:m,j} - \mathbf{V}_{1:m.,1:j-1}$work
24: $T_{1:j-1,j} = \mathbf{V}^T_{1:m,1:j-1}\mathbf{V}_{1:m,j}$
25: $T_{1:j-1,j} = -\tau_j * T_{1:j-1,j}$
26: $T_{1:j-1,j} = T_{1:j-1,1:j-1} * T_{1:j-1,j}$
27: **end if**
28: **end for**

Note: The workspace (Work), one can use $T_{1:j-1,j}$ as the workspace needed per iteration

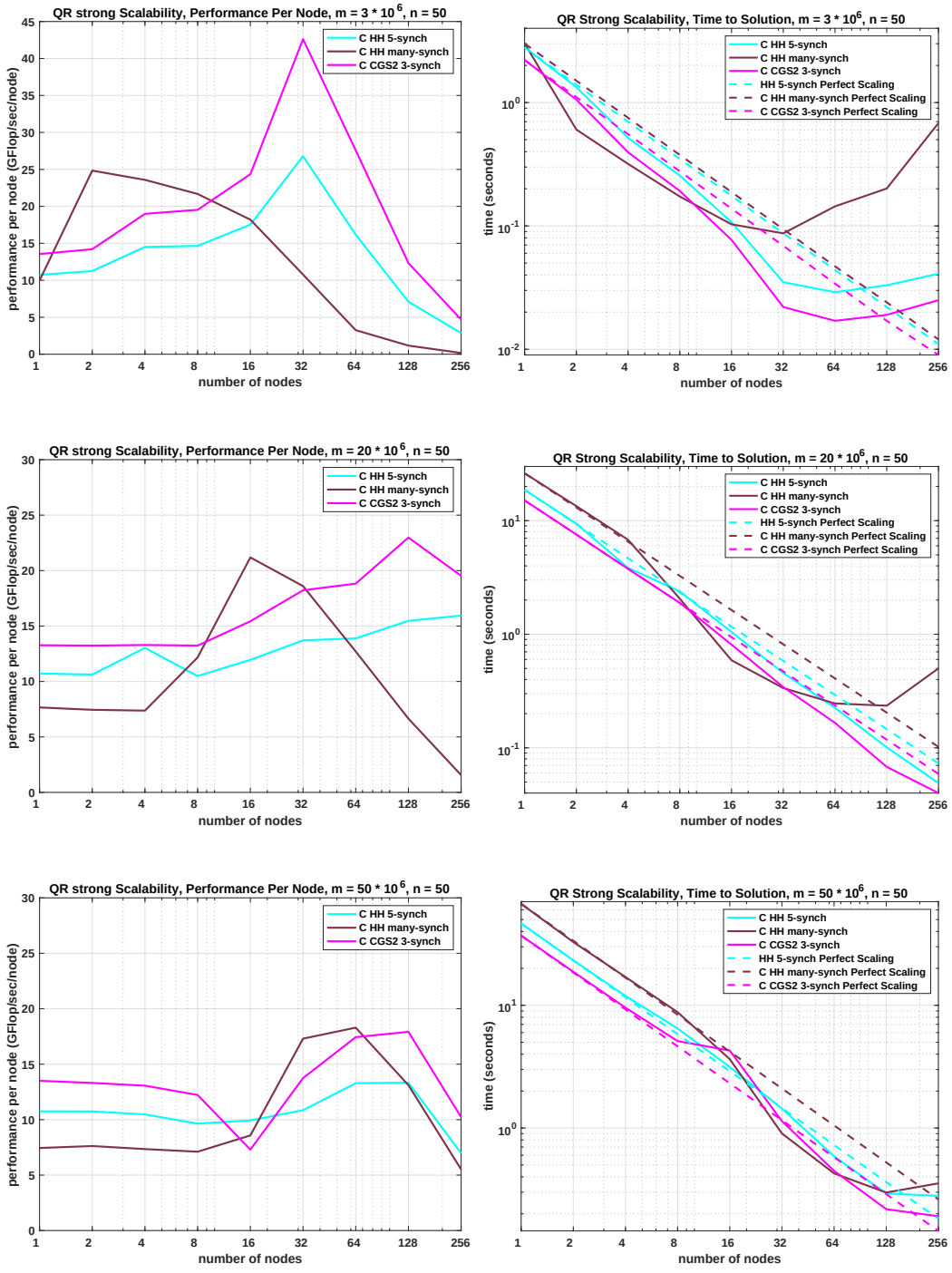


Fig. 5.1: Strong scalability experiment with number of rows varied, $m = 3,000,000$ (top), $m = 20,000,000$ (middle), $m = 50,000,000$ (bottom).

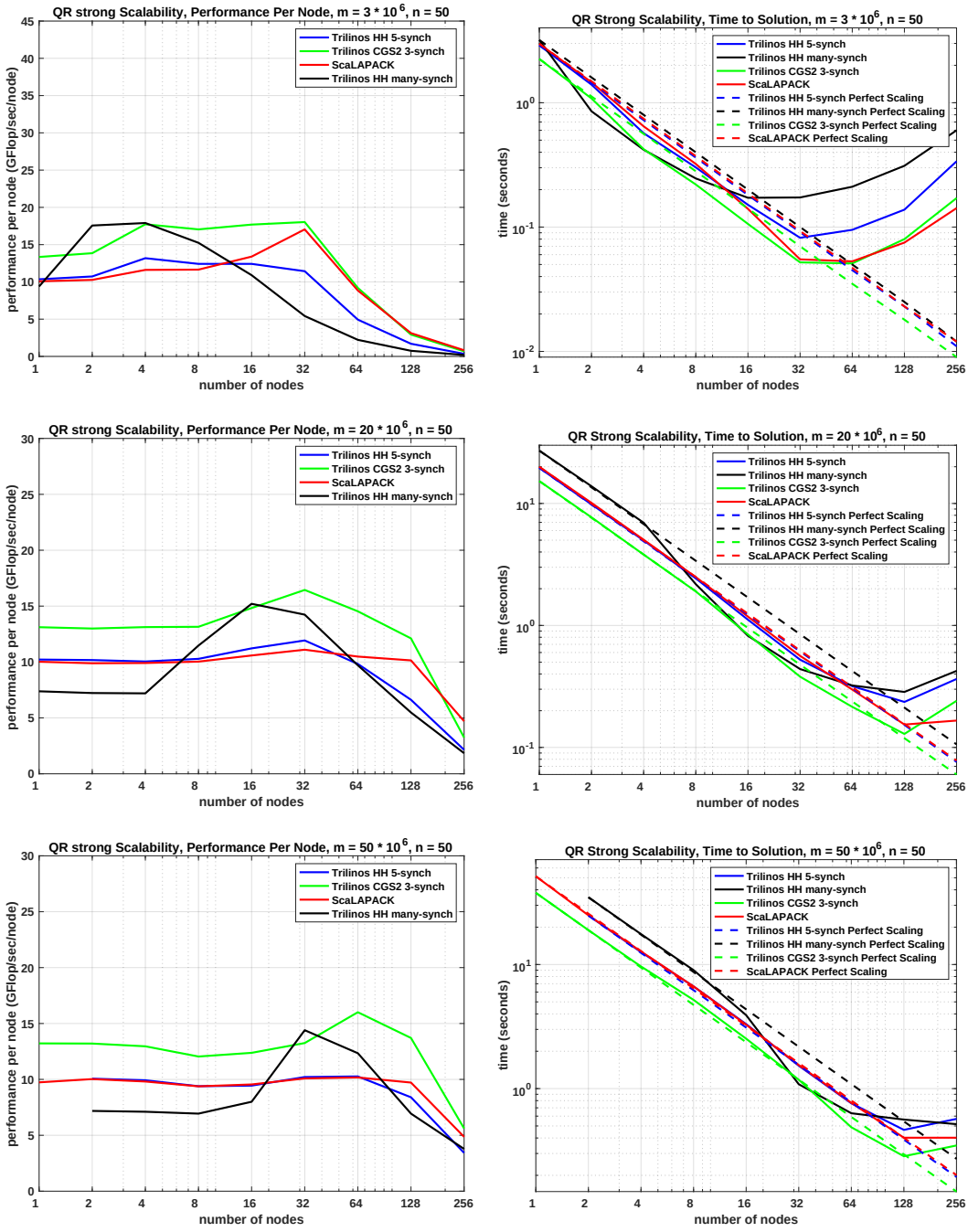


Fig. 5.2: Strong scalability experiment with number of rows varied, $m = 3,000,000$ (top), $m = 20,000,000$ (middle), $m = 50,000,000$ (bottom).

```
%
  clear
%
  m = 15;
  n = 5;
  log10KA = 10;
%
  U = randn(m,n); [U,~]=qr(U,0);
  V = randn(n,n); [V,~]=qr(V,0);
  S = diag( 10.^( linspace( 0, log10KA, n ) ) );
  A = U * S * V';
  clear U S V;
%
  Q   = zeros(m,n);
  R   = zeros(n,n);
  V   = zeros(m,n);
  T   = zeros(n,n);
  tau = zeros(n,1);
%
  for j = 1:n,
%
%     [ Q(1:m,j), R(1:j,j) ] = ...
%     orth_cgs2( Q(1:m,1:j-1), A(1:m,j) );
%
%     [ V(1:m,j), tau, R(1:j,j), Q(1:m,j) ] = ...
%     orth_hh_lvl1( V(1:m,1:j-1), tau, A(1:m,j) );
%
%     [ V(1:m,j), R(1:j,j), T(1:j,j), Q(1:m,j) ] = ...
%     orth_hh_lvl2( V(1:m,1:j-1), T(1:j-1,1:j-1), A(1:m,j) );
%
  end
%
  orth = norm(eye(n) - Q'*Q, 'fro');
  repres = norm( A - Q * R, 'fro') / norm( A, 'fro' );
  fprintf('|| I - Q''* Q ||           = %6.1e\n', orth );
  fprintf('|| A - Q * R || / || A || = %6.1e\n', repres );
%
```

Fig. B.1: Main Driver for the Following Functions

```
%
  function [ v, r, t, q ] = orth_hh_lvl2( V, T, a )
%
     m = size(V,1);
     j = size(V,2)+1;
%
     q = zeros(m,1);
     r = zeros(j,1);
     t = zeros(j,1);
%
     if (j > 1 )
     r(1:j-1,1) = V(1:m,1:j-1)' * a(1:m,1);          %%%%%%%%%% <- 1 synchronization
     r(1:j-1,1) = T(1:j-1,1:j-1)' * r(1:j-1,1);
     a(1:m,1) = a(1:m,1) - V(1:m,1:j-1) * r(1:j-1,1);
%
     r(1:j-1,1) = a(1:j-1,1);                        %%%%%%%%%% <- 1 broadcast
     end
%
     normx = norm( a(j+1:m,1) , 2);                  %%%%%%%%%% <- 2 synchronizations
     norma = sqrt( normx*normx +  a(j,1)*a(j,1) );
     if ( a(j,1) > 0.0e+00 ) v(j,1) = a(j,1)+norma; else v(j,1) = a(j,1)-norma; end
     v(j+1:m,1) = a(j+1:m,1) / v(j,1);
     if ( a(j,1) > 0.0e+00 ) r(j,1) = - norma; else r(j,1) = norma; end
     tau = 2.0e+00 / ( 1.0e+00 + ( normx / v(j,1) )^2 );
     v(j,1) = 1.0e+00;
     v(1:j-1,1) = 0.0e+00;
%
     q(j,1) = - tau;
     q(j+1:m,1) = v(j+1:m,1) * q(j,1) ;
     q(j,1) = 1.0e+00 + q(j,1) ;
     if (j > 1 )
     q(1:j-1,1) = 0.e+00;
     t(1:j-1,1) = V(1:m,1:j-1)' * q(1:m,1) ;         %%%%%%%%%% <- 3 synchronizations
     t(1:j-1,1) = T(1:j-1,1:j-1) * t(1:j-1,1) ;
     q(1:m,1) = q(1:m,1) - V(1:m,1:j-1) * t(1:j-1,1) ;
     end
%
     if (j > 1 )
     t(1:j-1,1) = V(1:m,1:j-1)' * v(1:m,1);          %%%%%%%%%% <- 4 synchronizations
     t(1:j-1,1) = -t(1:j-1,1) * tau;
     t(1:j-1,1) = T(1:j-1,1:j-1) * t(1:j-1,1);
     end
     t(j,1) = tau;
%
  end
%
```

Fig. B.2: Householder Level 2 Orthogonalization implemented within Matlab

```
%
  function [ a, tau, r, q ] = orth_hh_lvl1( V, tau, a )
%
     m = size(V,1);
     j = size(V,2)+1;
%
     r = zeros(j,1);
     q = zeros(m,1);
%
     for i = 1:j-1,
        alpha = tau(i) * ( V(i:m,i)' * a(i:m,1) ); %%%%%%%%%% <- 1 synchronization
        a(i:m,1) = a(i:m,1) - V(i:m,i) * alpha;    %%%%%%%%%%    j-1 each call
     end
%
     if (j>1), r(1:j-1,1) = a(1:j-1,1); end;       %%%%%%%%%% <- broadcast here
%
     normx = norm( a(j+1:m,1), 2);                 %%%%%%%%%% <- 1 synchronization
     norma = sqrt( normx*normx +  a(j,1)*a(j,1) );
     if ( a(j,1) > 0.0e+00 ) r(j,1) = - norma; else r(j,1) = norma; end
     if ( a(j,1) > 0.0e+00 ) a(j,1) = a(j,1)+norma; else a(j,1) = a(j,1)-norma; end
     a(j+1:m,1) = a(j+1:m,1) / a(j,1);
     tau(j) = 2.0e+00 / ( 1.0e+00 + ( normx / a(j,1) )^2 );
     a(1:j-1,1) = 0.0e+00;
     a(j,1) = 1.0e+00;
%
     q(j,1) = - tau(j);
     q(j+1:m,1) = a(j+1:m,1) * q(j,1);
     q(j,1) = 1.0e+00 + q(j,1);
     for i = j-1:-1:1,
       q(i,1) = - tau(i) * ( V(i+1:m,i)' * q(i+1:m,1) ); %%%%%% <- 1 synchronization
       q(i+1:m,1) = q(i+1:m,1) + V(i+1:m,i) * q(i,1);    %%%%%%    j-1 each call
     end
%
  end
```

Fig. B.3: Householder Level 1 Orthogonalization implemented within Matlab

```
%
  function [ q, r ] = orth_cgs2( Q, a )
%
     m = size(Q,1);
     j = size(Q,2)+1;
%
     r = zeros(j,1);
     h = zeros(j,1);
     q = zeros(m,1);
%
     r(1:j-1,1) = Q(1:m,1:j-1)' * a(1:m,1);     %%%%%%%%%% <- 1st synchronization
     q(1:m,1) = a(1:m,1) - Q(1:m,1:j-1) * r(1:j-1,1);
%
     h(1:j-1,1) = Q(1:m,1:j-1)' * q(1:m,1);     %%%%%%%%%% <- 2nd synchronization
     q(1:m,1) = q(1:m,1) - Q(1:m,1:j-1) * h(1:j-1,1);
%
     r(1:j-1,1) = r(1:j-1,1) + h(1:j-1,1);
%
     r(j,1) = norm( q(1:m,1) );                 %%%%%%%%%% <- 3rd synchronization
     q(1:m,1) = q(1:m,1) / r(j,1);
%
  end
%
```

Fig. B.4: Classical Gram-Schmidt Re-Orthogonalized implemented within Matlab

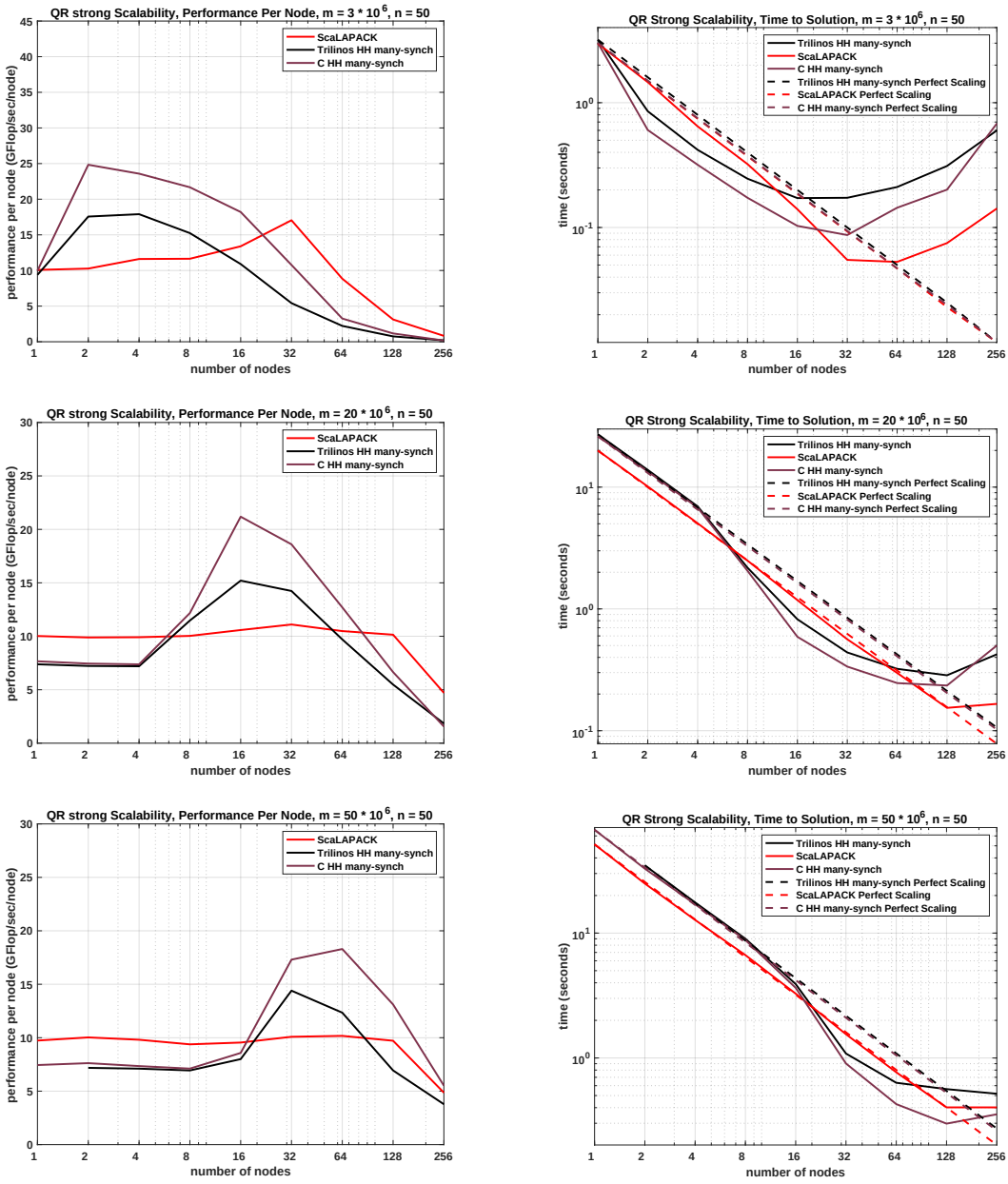


Fig. C.1: Scalability and performance plots using Householder many-synch orthogonalization scheme.

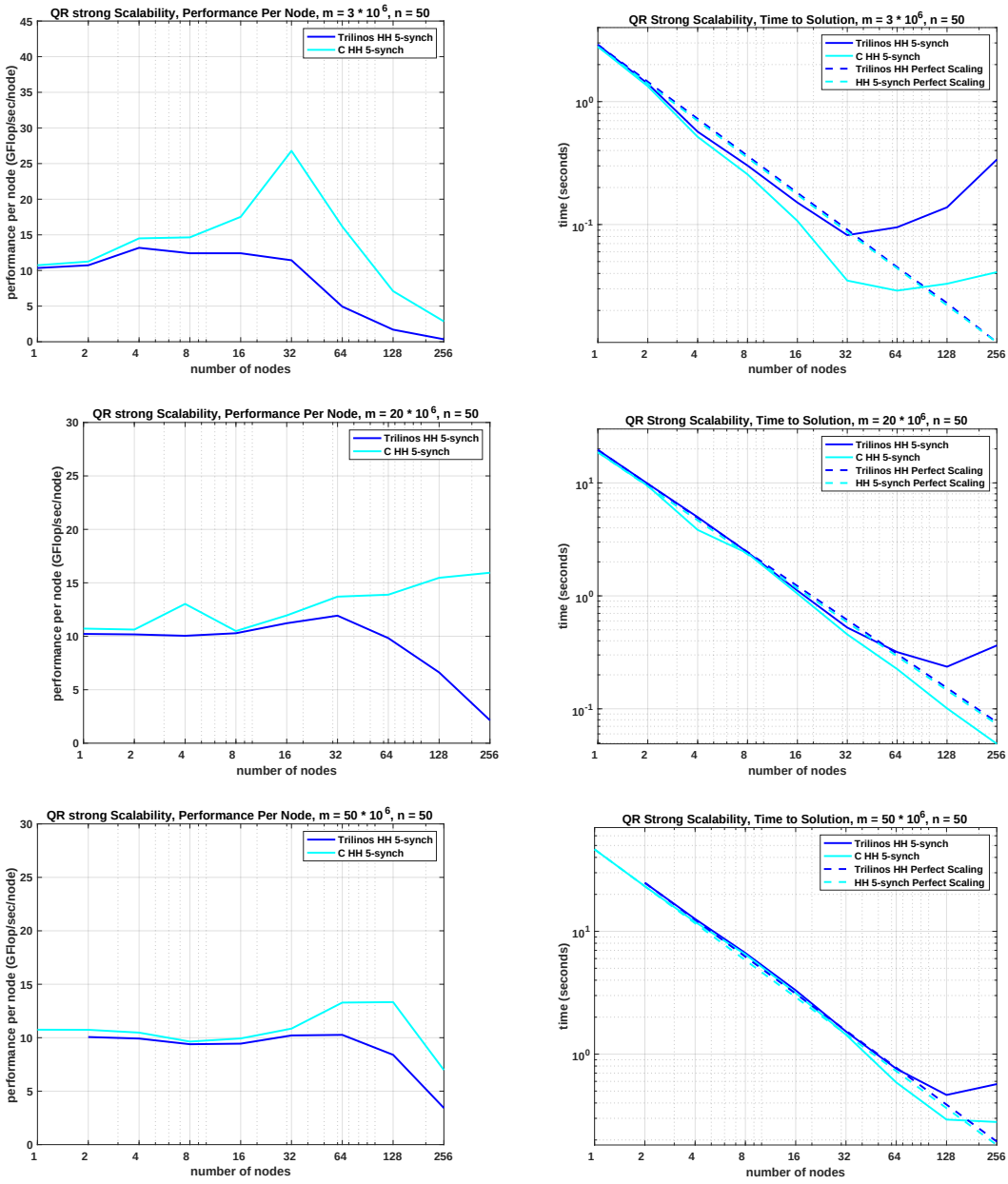


Fig. C.2: Scalability and performance plots using Householder five-synch orthogonalization scheme.

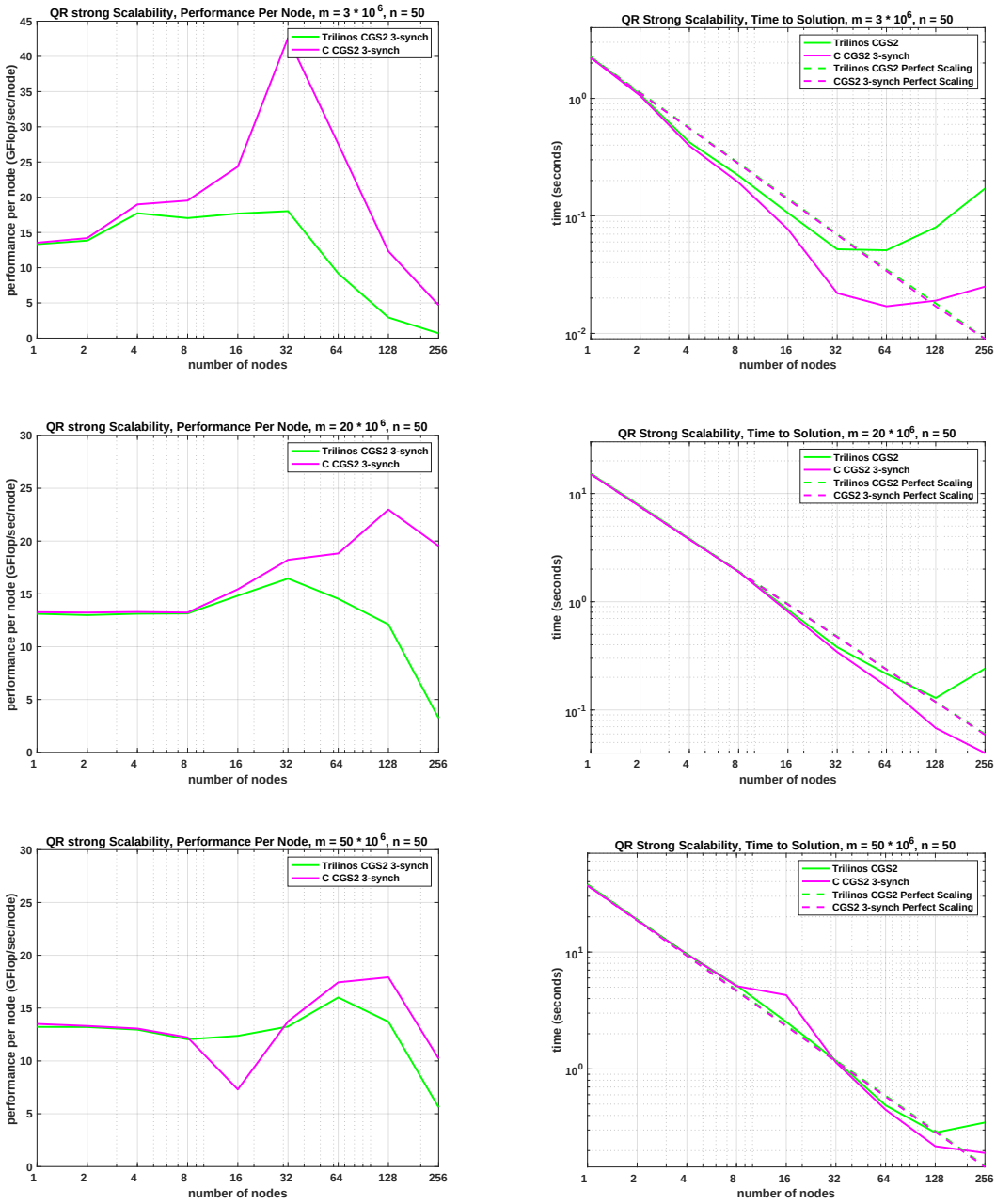


Fig. C.3: Scalability and performance plots using CGS2 three-synch orthogonalization scheme.

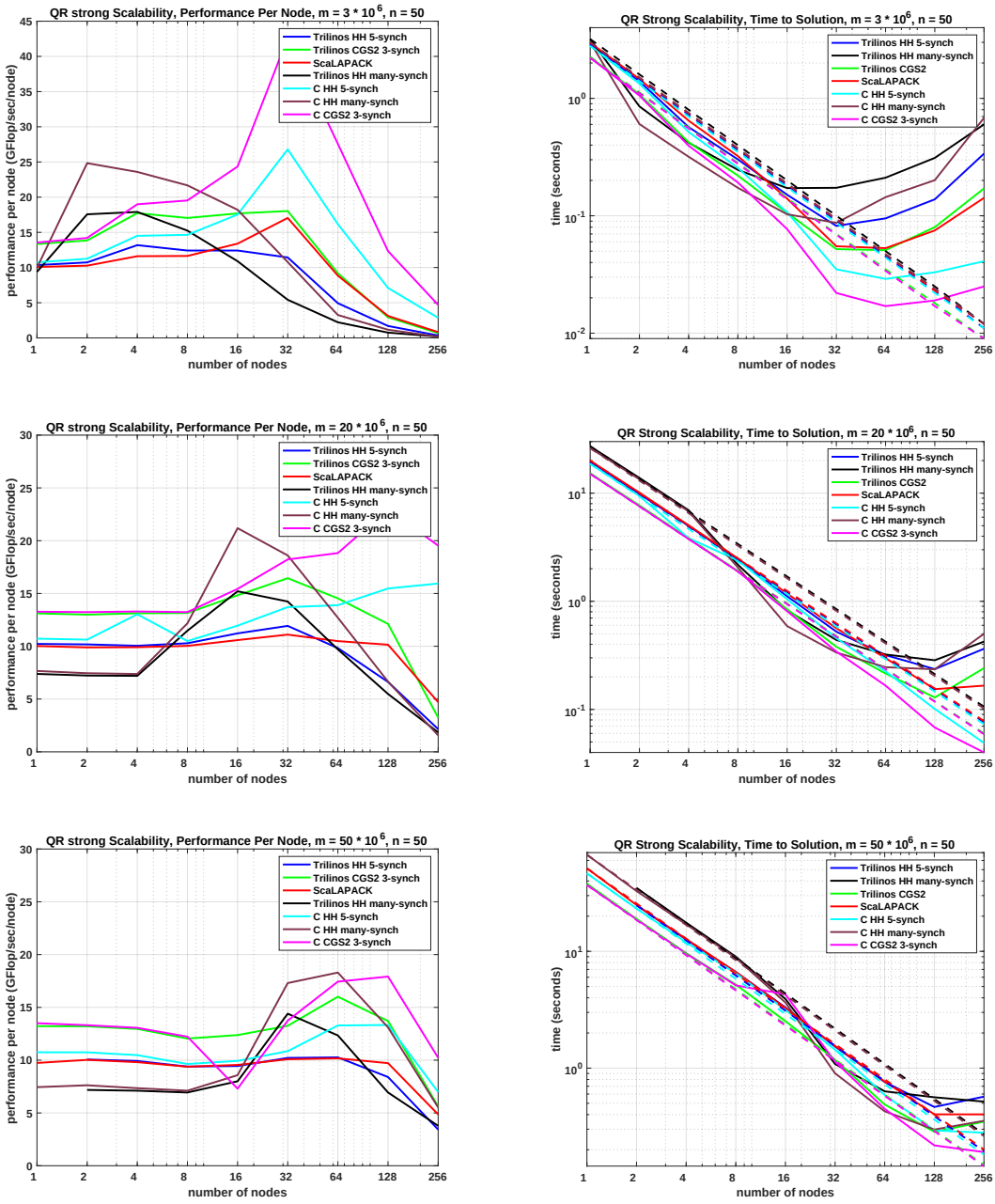


Fig. C.4: Scalability and performance plots using all orthognoalization schemes. Note - dashed lines represent perfect scalability of each method. We remove them from the legend to not overcrowd it, each perfect scalability line matches the color of its recorded data.

Algorithm 2 Householder Orthogonalization Level One (HH LVL1) Algorithm.

for $(j = 1; j <= n; j++)$ **do**
  $\mathbf{v} = \mathbf{a}_j$
  for $(i = 1; i <= j-1; i++)$ **do**
    work $= \mathbf{V}_{1:m,i}^T \mathbf{v}$
    work $=$ work $* \tau_i$
    $\mathbf{v} = \mathbf{v} - \mathbf{V}_{1:m,i} *$ work
  end for
  $\alpha = \mathbf{v}_{j+1:m} \mathbf{v}_{j+1:m}^T$
  $\beta = \sqrt{\alpha + \mathbf{v}_j^2}$
  $\gamma = \mathbf{v}_j + \beta * \text{sign}(\mathbf{v}_j)$
  $\tau_j = \frac{2}{1+(\alpha/\gamma^2)}$
  $R_{1:j-1,j} = \mathbf{v}_{1:j-1}$
  $R_{j,j} = -\beta * \text{sign}(\gamma)$
  $\mathbf{V}_{j+1:m,j} = \frac{1}{\gamma} \mathbf{v}_{j+1:m}$
  $\mathbf{V}_{j,j} = 1.0$
  $\mathbf{V}_{1:j-1,j} = 0.0$
  $\mathbf{Q}_{j+1:m,j} = -\mathbf{v}_{j+1:m} * \tau_j$
  $\mathbf{Q}_{j,j} = 1.0 - \tau_j$
  $\mathbf{Q}_{1:j-1,j} = 0.0$
  for $(i = j-1; i >= 1; i--)$ **do**
    work $= \mathbf{V}_{1:m,i}^T \mathbf{Q}_{1:m,j}$
    work $=$ work $* \tau_i$
    $\mathbf{Q}_{1:m,j} = \mathbf{Q}_{1:m,j} - \mathbf{V}_{1:m,i} *$ work
  end for
end for

Algorithm 3 Classical Gram Schmidt Re-Orthogonalized (CGS2) Algorithm.

if $(j == 1)$ **then**
  $r_{1,1} = \|\mathbf{a}_1\|$
  $\mathbf{Q}_1 = \frac{\mathbf{a}_1}{r_{1,1}}$
end if
for $(j = 2; j <= n; j++)$ **do**
  $h_{1:j-1,j} = \mathbf{Q}_{j-1}^T \mathbf{a}_j$
  $\mathbf{w}_j = \mathbf{a}_j - \mathbf{Q}_{j-1} h_{1:j-1,j}$
  $t_{1:j-1,j} = \mathbf{Q}_{1:j-1}^T \mathbf{w}_j$
  $\mathbf{u}_j = \mathbf{w}_j - \mathbf{Q}_{j-1} t_{1:j-1,j}$
  $r_{1:j-1,j} = h_{1:j-1,j} + t_{1:j-1,j}$
  $r_{j,j} = ||\mathbf{u}_j||$
  $\mathbf{q}_j = \frac{\mathbf{u}_j}{r_{j,j}}$
end for

DISTRIBUTED MEMORY GRAPH COLORING ALGORITHMS ON MULTIPLE GPUS

IAN BOGLE*, ERIK G. BOMAN†, KAREN DEVINE‡, SIVASANKARAN RAJAMANICKAM§, AND GEORGE M. SLOTA¶

Abstract. Graph coloring is often used in parallelizing scientific computations that run in distributed and multi-GPU environments; it identifies sets of independent data that can be updated in parallel. Many algorithms exist for graph coloring on a single GPU or in distributed memory, but hybrid MPI+GPU algorithms have been unexplored until this work, to the best of our knowledge. We present several MPI+GPU coloring approaches that use implementations of the distributed coloring algorithms of Gebremedhin et al. and the shared-memory algorithms of Deveci et al. The on-node parallel coloring uses implementations in KokkosKernels, which provide parallelization for both multicore CPUs and GPUs. We further extend our approaches to solve for distance-2 coloring, giving the first known distributed and multi-GPU algorithm for this problem. In addition, we propose novel methods to reduce communication in distributed graph coloring. Our experiments show that our approaches operate efficiently on inputs too large to fit on a single GPU and scale up to graphs with 76.7 billion edges running on 128 GPUs.

1. Introduction. This work will appear in the proceedings of the IAˆ3 workshop at SC20 [4].We present new multi-GPU, distributed memory implementations of distance-1 and distance-2 graph coloring. *Distance-1 graph coloring* assigns *colors* (i.e., labels) to all vertices in a graph such that no two neighboring vertices have the same color. Similarly, *distance-2 coloring* assigns colors such that no vertices within *two hops*, also called a "two-hop neighborhood," have the same color. Usually, these problems are formulated as NP-hard optimization problems, where the number of colors used to fully color a graph is minimized. Serial heuristic algorithms have traditionally been used to solve these problems, one of the most notable being the DSatur algorithm of Brélaz [7]. More recently, parallel algorithms [11, 6] have been proposed; usually requiring multiple *rounds* to correct for improper *speculative* colorings produced in multi-threaded or distributed environments.

There are many useful applications of graph coloring. Most commonly, it is employed to find concurrency in parallel scientific computations [11, 3]; all data sharing a color can be updated in parallel without incurring race conditions. Other applications use coloring as a preprocessing step to speed up the computation of Jacobian and Hessian matrices [16] and to identify short circuits in printed circuit designs [14]. Despite the intractability of minimizing the number of colors for non-trivial graphs, such applications benefit from good heuristic algorithms that produce small numbers of colors. For instance, Deveci et al. [11] show that a smaller number of colors used by a coloring-based preconditioner reduces the runtime of a conjugate gradient solver by 33%.

In particular, this work is motivated by the use of graph coloring as a preprocessing step for distributed scientific computations such as automatic differentiation [17]. For such applications, assembling the associated graphs on a single node to run a sequential coloring algorithm may not be feasible [6]. As such, we focus on running our algorithms on the parallel architectures used by the underlying applications. These architectures typically are highly distributed, with multiple CPUs and/or GPUs per node. Therefore, we specifically consider coloring algorithms that can use the "MPI+X" paradigm, where "X" is multicore CPU or GPU acceleration.

*Rensselaer Polytechnic Institute, boglei@rpi.edu
†Sandia National Laboratories, egboman@sandia.gov
‡Sandia National Laboratories,kddevin@sandia.gov
§Sandia National Laboratories, srajama@sandia.gov
¶Rensselaer Polytechnic Institute, slotag@rpi.edu

1.1. Contributions. We present and examine two MPI+X implementations of distance-1 coloring as well as one MPI+X implementation of distance-2 coloring. In order to run on a wide variety of architectures, we use the Kokkos performance portability framework [13, 1] for on-node parallelism and Trilinos [19] for distributed MPI-based parallelism. The combination of Kokkos and MPI allows our algorithms to run on multiple multicore CPUs or multiple GPUs in a system. However, for this paper, we focus on the performance of our algorithms in MPI+GPU environments. For distance-1 coloring of real-world networks, our algorithms see up to 28x speedup on 128 GPUs compared to a single GPU, and only a 7.5% increase in colors on average. For distance-2 coloring, our algorithm also sees up to 28x speedup, and a 4.9% increase in colors in the *worst* case. We also demonstrate good weak scaling behavior on up to 128 GPUs on graphs with up to 12.8 billion vertices and 76.7 billion edges in size.

2. Background.

2.1. Coloring Problem . While there exist many definitions of the "graph coloring problem," we specifically consider variants of distance-1 and distance-2 coloring. Consider graph $G = (V, E)$ with vertex set V and edge set E. *Distance-1 coloring* assigns to each vertex $v \in V$ a color $C(v)$ such that $\forall (u, v) \in E, C(u) \neq C(v)$. In *distance-2 coloring*, colors are assigned such that $\forall (u, v), (v, w) \in E, C(u) \neq C(v) \neq C(w)$; i.e., all vertices within two hops of each other have different colors. When a coloring satisfies one of the above constraints, it is called *proper*. The goal is to find proper colorings of G such that the total number of different colors used is minimized.

2.2. Coloring Background. While minimizing the number of colors is NP-hard, serial coloring algorithms using greedy heuristics have been effective for many applications [15]. The serial greedy algorithm in Algorithm 1 colors vertices one at a time. Colors are represented by integers, and the smallest usable color is assigned as a vertex's color. Most serial and parallel coloring algorithms use some variation of greedy coloring, with algorithmic differences usually involving the processing order of vertices or, in parallel, the handling of conflicts and communication.

Algorithm 1 Serial greedy coloring algorithm

```
procedure SerialGreedy(Graph G = (V, E))
    C(∀v ∈ V) ← 0                          ▷ Initialize all colors as null
    for all v ∈ V in some order do
        c ← the smallest color not used by a neighbor of v
        C(v) ← c
```

Conflicts in a coloring are edges that violate the color-assignment criterion; for example, in distance-1 coloring, a conflict is an edge with both endpoints sharing the same color. Colorings that contain conflicts are not proper colorings, and are referred to as *pseudo-colorings*. Pseudo-colorings arise only in parallel coloring, as conflicts arise only when two vertices are colored concurrently. A coloring's "quality" refers to the number of colors used; higher quality colorings of a graph G use fewer colors, while lower quality colorings of G use more colors.

2.3. Parallel Coloring Algorithms. There are two popular approaches to parallel graph coloring. The first concurrently finds independent sets of vertices and concurrently colors all of the vertices in each set; this approach was used by Jones and Plassmann [21]. Osama et al. [23] implement approaches based on finding independent sets on a single GPU and explore the impact of varying the baseline independent set algorithm.

The second approach, referred to as "speculate and iterate" [9], colors as many vertices as possible in parallel and then iteratively fixes conflicts in the resulting pseudo-coloring until no conflicts remain. Çatalyürek et al. [9] and Rokos et al. [24] present shared-memory implementations based on the speculate and iterate approach. Deveci et al. [11] present implementations based on the speculate and iterate approach that are scalable on GPUs. Distributed-memory algorithms such as those in [6, 26] use the speculate and iterate approach. Grosset et al. [18] present a hybrid speculate and iterate approach that splits computations between the CPU and a single GPU, but does not operate on multiple GPUs in a distributed memory context. Bozdağ et al. [6] showed that, in distributed memory, the speculative approach is more scalable than methods based on the independent set approach of Jones and Plassmann. As such, we choose a speculative and iterative approach with our algorithms.

2.4. Distributed Coloring. In a typical distributed memory setting, an input graph is split into subgraphs that are assigned to separate processes. A process's *local graph* $G_l = \{V_l + V_g, E_l + E_g\}$ is the subgraph assigned to the process. Its vertex set V_l contains *local vertices*, and a process is said to *own* its local vertices. The intersection of all processes' V_l is null, and the union equals V. The local graph also has non-local vertex set V_g, with such non-local vertices commonly referred to as *ghost vertices*; these vertices are copies of vertices owned by other processes. To ensure a proper coloring, each process needs to store color state information for both local vertices and ghost vertices; typically, ghost vertices are treated as read-only. The local graph contains edge set E_l, edges between local vertices, and E_g, edges containing at least one ghost vertex as an endpoint. Bozdağ et al. [6] also defines two subsets of local vertices: *boundary vertices* and *interior vertices*. Boundary vertices are locally owned vertices that share an edge with at least one ghost; interior vertices are locally owned vertices that do not neighbor ghosts. For processes to communicate colors associated with their local vertices, each vertex has a unique global identifier (GID).

3. Methods. We present three hybrid MPI+GPU algorithms, called Distance-1 (D1), Distance-1 Two Ghost Layer (D1-2GL) and Distance-2 (D2). D1 and D1-2GL solve the distance-1 coloring problem and D2 does distance-2 coloring. We leverage Trilinos [19] for distributed MPI-based parallelism and Kokkos [13] for on-node parallelism. KokkosKernels [1] provides baseline implementations of distance-1 and distance-2 coloring algorithms that we use and modify for our local coloring and recoloring subroutines.

Our three proposed algorithms follow the same basic framework, which builds upon that of Bozdağ et al. [6]. Bozdağ et al. observe that interior vertices can be properly colored independently on each process without creating conflicts or requiring communication. They propose first coloring interior vertices, and then coloring boundary vertices in small batches over multiple rounds involving communication between processes. This approach can reduce the occurrence of conflicts, which in turn reduces the amount of communication necessary to properly color the boundary. In our approach, we color all *local* vertices first. Then we fix all conflicts after communication of boundary vertices' colors. Several rounds of conflict resolution and communication may be needed to resolve all conflicts. We found that this approach was generally faster than the batched boundary coloring, and it allowed us to use existing parallel coloring routines in KokkosKernels without substantial modification.

Algorithm 2 demonstrates the general approach for our three speculative distributed algorithms. First, each process colors all local vertices with a shared-memory algorithm. Then, each process communicates its boundary vertices' colors to processes with corresponding ghosts. Processes detect conflicts in a globally consistent way and remove the colors of conflicted vertices. Finally, processes locally recolor all uncolored vertices, communicate updates, detect conflicts, and repeat until no conflicts are found.

Algorithm 2 Distributed-Memory Speculative Coloring

```
procedure Parallel-Color(Graph G = (V, E))
    Color all local vertices
    Communicate colors of boundary vertices
    do
        Detect conflicts
        Recolor conflicting vertices
        Communicate updated boundary colors
    while Conflicts exist
```

3.1. Distance-1 Coloring (D1). Our D1 method begins by independently coloring all owned vertices on each process using the GPU-enabled algorithms by Deveci et al. [11] VB_BIT and EB_BIT in KokkosKernels [1]. VB_BIT uses vertex-based parallelism; each vertex is colored by a single thread. VB_BIT uses compact bit-based representations of colors to make it performant on GPUs. EB_BIT uses edge-based parallelism; a thread colors the endpoints of a single edge. EB_BIT also uses the compact color representation to reduce memory usage on GPUs.

For graphs with skewed degree distribution (e.g., social networks), edge-based parallelism typically yields better workload balance between GPU threads. We observed that for graphs with a sufficiently large maximum degree, edge-based EB_BIT outperformed vertex-based VB_BIT on Tesla V100 GPUs. Therefore, we use a simple heuristic based on maximum degree: we use EB_BIT for graphs with maximum degree greater than 6000; otherwise, we use VB_BIT.

Algorithm 3 Algorithm to identify and resolve conflicts

```
procedure Check-Conflicts(v, u, colors, GID)
    conflict ← 0
    if colors[v] = colors[u] then
        if rand(GID[v]) > rand(GID[u]) then
            colors[v] ← 0
        else if rand(GID[u]) > rand(GID[v]) then
            colors[u] ← 0
        else
            if GID[v] > GID[n] then
                colors[v] ← 0
            else
                colors[u] ← 0
        conflict ← 1
    return conflict
```

Algorithm 4 shows the conflict-resolution inner loop of Algorithm 2. This algorithm runs on each process using its owned local graph G_l. It detects conflicts across processor boundaries and recolors vertices to resolve the conflicts.

After the initial coloring, only boundary vertices can be in conflict with one another[1]. We perform a full exchange of boundary vertices' colors using Trilinos [19]. Specifically, we

[1] As suggested by Bozdağ et al., we considered reordering local vertices to group all boundary vertices together for ease of processing. This optimization did not show benefit in our implementation, as reordering tended to be slower than coloring of the entire local graph.

Algorithm 4 Distance-1 conflict resolution and recoloring

```
procedure Resolve-Conflicts(
  Local Graph G_l = {V_l + V_g, E_l + E_g}, colors, GID)
  conflicts ← 0
  for all v ∈ V_g do in parallel
    for all ⟨v, u⟩ ∈ (E_g) do
      conflicts ← conflicts + Check-Conflicts(v, u, ...)
      if colors[v] = 0 then
        break
  Allreduce(conflicts, SUM)                          ▷ Get global conflicts
  gc ← current colors of all ghosts
  if conflicts > 0 then
    colors = Color(G_l, colors)                      ▷ Recolor vertices
    Replace ghost colors with gc
    Communicate recolored vertices to ghost copies
  return conflicts
```

use the FEMultiVector class of Tpetra [20] to communicate the colors of boundary vertices to their ghost copies on other processes via an all-to-all exchange. After the initial all-to-all exchange, we only communicate the colors of boundary vertices which have been recolored. After each process receives its ghosts' colors, it detects conflicts by checking each owned vertex's color against the colors of its neighbor as in Algorithm 4. The conflict detection is done in parallel over the owned vertices using Kokkos. The overall time of conflict detection is small enough that any imbalance resulting from our use of vertex-based parallelism is insignificant relative to end-to-end times for the D1 algorithm.

When a conflict is found, only one vertex involved in the conflict needs to be recolored. Since conflicts happen on edges between two processes' vertices, both processes must agree on which vertex will be recolored. We adopt the random conflict resolution scheme of Bozdağ et al. We use a random number generator (given as the "rand" function in Algorithm 3) seeded by the GID of each conflicted vertex, as this produces a consistent set of random numbers across processes without communication. In a conflict, the vertex with the larger random number is chosen for recoloring. For the rare case in which both random numbers are equal, the tie is broken based on GID. Using random numbers instead of simply using GIDs helps balance recoloring workload across processes.

Once we have identified all conflicts, we again use VB_BIT or EB_BIT to recolor the determined set of conflicting vertices. We modified KokkosKernels' coloring implementations to accept a "partial" coloring and the full local graph, including ghosts. (Our initial coloring phase did not need ghost information.) We also modified VB_BIT to accept a list of vertices to be recolored. Such a modification was not feasible for EB_BIT.

Before we detect conflicts and recolor vertices, we save a copy of the ghosts' colors (gc in Algorithm 4). Then we give color zero to all vertices that will be recolored; our coloring functions interpret color zero as uncolored. To prevent the coloring functions from resolving conflicts without respecting our conflict resolution rules (thus preventing convergence of our parallel coloring), we allow a process to temporarily recolor some ghosts, even though the process does not have enough color information to correctly recolor them. The ghosts' colors are then restored to their original values in order to keep ghosts' colors consistent with their owning process. Then, we communicate only recolored owned vertices, ensuring that recoloring changes only owned vertices.

Table 3.1: Summary of input graphs. δ_{avg} refers to average degree and δ_{max} refers to maximum degree. Numeric values listed are after preprocessing to remove multi-edges and self-loops. k = thousand, M = million, B = billion.

Graph	Class	#Vertices	#Edges	δ_{avg}	δ_{max}	Memory (GB)
ldoor	PDE Problem	0.9 M	21 M	45	77	0.32
Audikw_1	PDE Problem	0.9 M	39 M	81	345	0.59
Bump_2911	PDE Problem	2.9 M	63 M	43	194	0.96
Queen_4147	PDE Problem	4.1 M	163 M	78	89	2.5
soc-LiveJournal1	Social Network	4.8 M	43 M	18	20 k	0.67
hollywood-2009	Social Network	1.1 M	57 M	99	12 k	0.86
twitter7	Social Network	42 M	1.4 B	35	2.9 M	21
com-Friendster	Social Network	66 M	1.8 B	55	5.2 k	27
europe_osm	Road Network	51 M	54 M	2.1	13	1.2
indochina-2004	Web Graph	7.4 M	194 M	26	256 k	2.9
MOLIERE_2016	Document Mining	30 M	3.3 B	80	2.1 M	49
rgg_n_2_24_s0	Synthetic Graph	17 M	133 M	15	40	2.1
kron_g500-logn21	Synthetic Graph	2.0 M	182 M	87	8.7	2.7
mycielskian19	Synthetic Graph	393 k	452 M	2.3 k	196 k	6.7
mycielskian20	Synthetic Graph	786 k	1.4 B	3.4 k	393 k	21

3.2. Two Ghost Layers Coloring (D1-2GL). Our second distance-1 coloring algorithm, D1-2GL, follows the D1 method, but adds another ghost vertex "layer" to the subgraphs on each process. In D1, a process' subgraph does not include neighbors of ghost vertices unless those neighbors are already owned by the process. In D1-2GL, we include all neighbors of ghost vertices (the two-hop neighborhood of local vertices) in each process's subgraph, giving us "two ghost layers." To the best of our knowledge, this approach has not been explored before with respect to graph coloring.

This method can reduce the total amount of communication relative to D1 for certain graphs by reducing the total number of recoloring rounds needed. In particular, for mesh or otherwise regular graphs, the second ghost layer is primarily made up of interior vertices on other processes. Interior vertices are never recolored, so the colors of the vertices in the second ghost layer are fixed. Each process can then directly resolve more conflicts in a consistent way, thus requiring fewer rounds of recoloring. Fewer recoloring rounds results in fewer collective communications.

However, in D1-2GL, each communication can be more expensive, because a larger boundary from each process is communicated. Also, in irregular graphs, the second ghost layer often does not have mostly interior vertices. The relative proportion of interior vertices in the second layer also gets smaller as the number of processes increases. For the extra ghost layer to pay off, it must reduce the number of rounds of communications enough to make up for the increased cost of each communication. We discuss this more in our results.

To construct the second ghost layer on each process, processes exchange the adjacency lists of their boundary vertices; this step is needed only once. After the ghosts' connectivity information is added, we use the same coloring approach as in D1. However, we optimize our conflict detection by looking through only the ghost vertices' adjacencies (E_g), as they neighbor all local boundary vertices. By keeping the new ghost adjacency information separate from the local graph, we can detect all conflicts by examining only the edges between ghosts and their neighbors.

3.3. Distance-2 Coloring (D2). Our distance-2 coloring algorithm, D2, builds upon both D1 and D1-2GL. As with distance-1 coloring, we use algorithms from Deveci et al. in KokkosKernels for local distance-2 coloring. Specifically, we use NB_BIT, which is a "net-based" distance-2 coloring algorithm that uses the approach described by Taş et al. [29]. Instead of checking for distance-2 conflicts only between a single vertex and its two-hop

neighborhood, the net-based approach detects distance-2 conflicts among the immediate neighbors of a vertex. Our D2 approach also utilizes a second ghost layer to give each process the full two-hop neighborhood of its boundary vertices. This enables each process to directly check for distance-2 conflicts with local adjacency information. To find a distance-2 conflict for a given vertex, its entire two-hop neighborhood must be checked for potential conflicting colors.

Algorithm 5 Distance-2 conflict detection

```
procedure DETECT-D2-CONFLICTS(
  Local Graph G_l = {V_l + V_g, E_l + E_g}, colors, GID)
  conflicts ← 0
  for all v ∈ V_l do in parallel
    for all ⟨v, u⟩ ∈ (E_l + E_g) do
      conflicts ← conflicts + Check-Conflicts(v, u, ...)
      if colors[v] = 0 then
        break
      for all ⟨u, x⟩ ∈ (E_l + E_g) do
                                  ▷ u is one hop and x is two hops from v
        conflicts ← conflicts + Check-Conflicts(v, x, ...)
        if colors[v] = 0 then
          break
      if colors[v] = 0 then
        break
  return conflicts
```

Algorithm 5 shows the straightforward way in which we detect conflicts in D2 for each process. We again use vertex-based parallelism while detecting conflicts; each thread examines the entire two-hop neighborhood of a vertex v. As with distance-1 conflict detection, we identify all local conflicts and use a random number generator to ensure that vertices to be recolored are chosen consistently across processes. The iterative recoloring method of D1 then also works for D2 – we recolor all conflicts, replace the old ghost colors, and then communicate local changes.

3.4. Partitioning. We assume that target applications partition and distribute their input graphs in some way before calling these coloring algorithms. In our experiments, we used XtraPuLP v0.3 [27] to partition our graphs. Determining optimal partitions for coloring is not our goal in this work. Rather, we have chosen a partitioning strategy representative of that used in many applications. We partition graphs by balancing the number of edges per-process and minimizing a global edge-cut metric. This approach effectively balances per-process workload and helps minimize global communication requirements.

4. Experimental Setup. We performed scaling experiments on the AiMOS supercomputer housed at Rensselaer Polytechnic Institute. The system has 268 nodes, each equipped with 2 IBM Power 9 processors clocked at 3.15 GHz, 4x NVIDIA Tesla V100 GPUs with 16 GB of memory connected via NVLink, 512 GB of RAM, and 1.6 TB Samsung NVMe Flash memory. Inter-node communications uses a Mellanox Infiniband interconnect. We compile with xlC 16.1.1 and use Spectrum MPI with GPU-Direct communication disabled.

The input graphs we used are listed in Table 3.1. We primarily used graphs from the SuiteSparse Matrix Collection [10]. The maximum degree, δ_{max}, can be considered an

upper bound for the number of colors used, as any incomplete, connected, and undirected graph can be colored using at most δ_{max} colors [8]. We selected many of the same graphs used by Deveci et al. to allow for direct performance comparisons. We include many graphs from Partial Differential Equation (PDE) problems because they are representative of graphs used with Automatic Differentiation [17], which is a target application for graph coloring algorithms. We also include social network graphs and a web crawl to demonstrate scaling of our methods on irregular real-world datasets. We preprocessed all graphs to remove multi-edges and self-loops, and we used subroutines from HPCGraph [28] for efficient I/O.

We compare our implementation against the distributed distance-1 and distance-2 coloring in the Zoltan [12] package of Trilinos. Zoltan's implementations are based directly on Bozdağ et al. [6]. Zoltan's distributed algorithm for distance-2 coloring requires only a single ghost layer, and to reduce conflicts, the boundary vertices are colored in small batches. For our results, we ran Zoltan and our approaches with four MPI ranks per node on AiMOS, and used the same partitioning method across all of our comparisons. Our methods D1, D1-2GL, and D2 were run with four GPUs and four MPI ranks (one per GPU) per node. Zoltan uses only MPI parallelism; it does not use GPU or multicore parallelism. For consistency, we set Zoltan to four MPI ranks per node, and use the same number of nodes for experiments with Zoltan and our methods. We used Zoltan's default coloring parameters; we did not experiment with options for vertex visit ordering, boundary coloring batch size, etc.

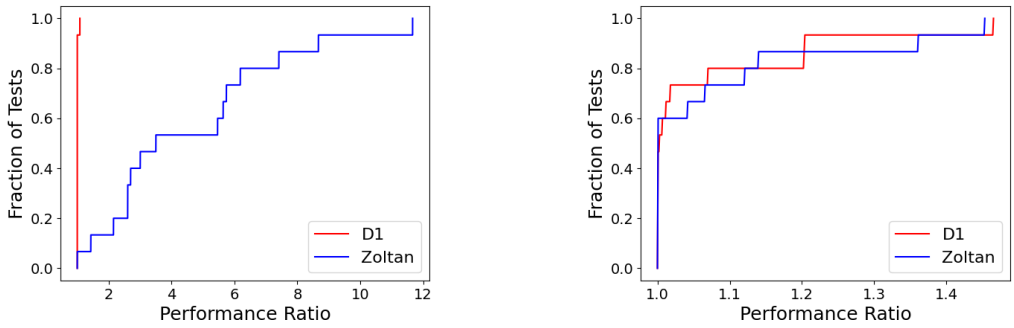
We omit direct comparison to single-node GPU coloring codes such as CuSPARSE [22], as we use subroutines for on-node coloring from Deveci et al. [11]. Deveci et al. have already performed a comprehensive comparison between their coloring methods and those in CuSPARSE, reporting an average speedup of 50% across a similar set of test instances. As such, we are confident that our on-node GPU coloring is representative of the current state-of-the-art.

5. Results. For our experiments, we compare overall performance for D1 and D2 on up to 128 ranks versus Zoltan. Our performance metrics include execution time, parallel scaling, and number of colors used. We do not include the partitioning time for XtraPuLP; we assume target applications will partition and distribute their graphs. Each of the results reported represents an average of five runs.

5.1. Distance-1 Performance. We summarize the performance of our algorithms relative to Zoltan using performance profiles. Performance profiles plot the proportion of problems an algorithm can solve for a given relative cost. The relative cost is obtained by dividing each approach's execution time (or colors used) by the better approach's execution time (or colors used) for a given problem. In these plots, the line that is higher represents the better performing algorithm. The further to the right that an algorithm's profile is, the worse it is relative to the other algorithm.

We ran D1 and Zoltan with 128 MPI ranks to color the 15 SuiteSparse graphs in Table 3.1. Some skewed graphs (e.g., twitter7) did not run on 128 ranks on Zoltan or D1; in those cases we use the largest run that completed for both approaches. D1 used MPI plus 128 Tesla V100 GPUs, while Zoltan used MPI on 128 Power9 CPU cores across 32 nodes (four MPI ranks per node). Figure 5.1(a) shows that D1 outperforms Zoltan in terms of execution time in these experiments. The D1 method is the fastest in roughly 95% of the cases; Zoltan outperforms D1 in only a single instance. D1 has at most a 11.6x speedup over Zoltan (with the europe_osm graph) and at worst an 8% slowdown relative to Zoltan (with Audikw_1).

Figure 5.1(b) shows that Zoltan outperforms D1 in terms of color usage. Zoltan uses fewer colors in over 60% of our experiments. However, in most cases, D1 uses no more than

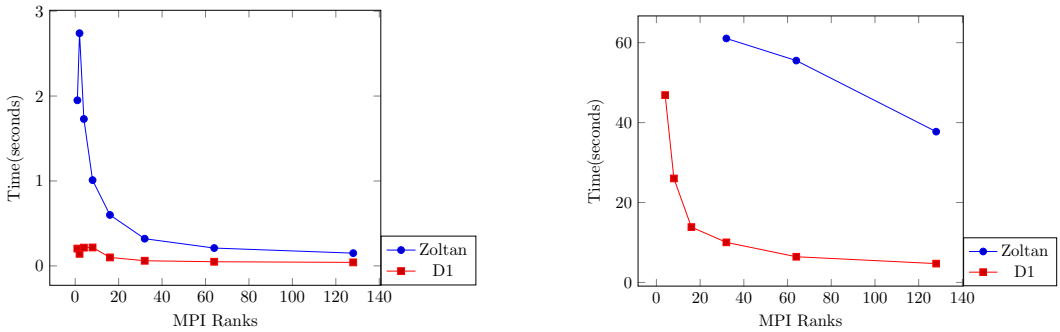


(a) Runtime performance profile (b) Color performance profile

Fig. 5.1: Performance profiles comparing D1 on 128 Tesla V100 GPUs with Zoltan's distance-1 coloring on 128 Power9 cores in terms of (a) execution time and (b) number of colors computed for the graphs listed in Table 3.1.

5% more colors than Zoltan. With the twitter7 graph, Zoltan uses 45% fewer colors than D1, but with Mycielskian20, D1 uses 41% fewer colors than Zoltan. On average, D1 uses 6.8% more colors than Zoltan. These increases in the number of colors exist because of the higher concurrency used by D1 relative to Zoltan.

5.2. Distance-1 Strong Scaling. Figure 5.2 shows strong scaling times for Queen_4147 and com-Friendster. These graphs are selected for presentation because they are the largest graphs for their respective problem domains. Data points that are absent were the result of out-of-memory issues or execution times (including partitioning) that were longer than our single job allocation limits. D1 scales better on the com-Friendster graph than on Queen_4147, as the GPUs can be more fully utilized with the much larger com-Friendster graph. For Queen_4147, D1 is at least 2.7x faster than Zoltan for each run, and D1 uses 12% fewer colors than Zoltan in the 128 rank run. For com-Friendster, D1 is roughly 8x faster than Zoltan in the 128 rank run, but D1 uses 26% more colors than Zoltan in that case.

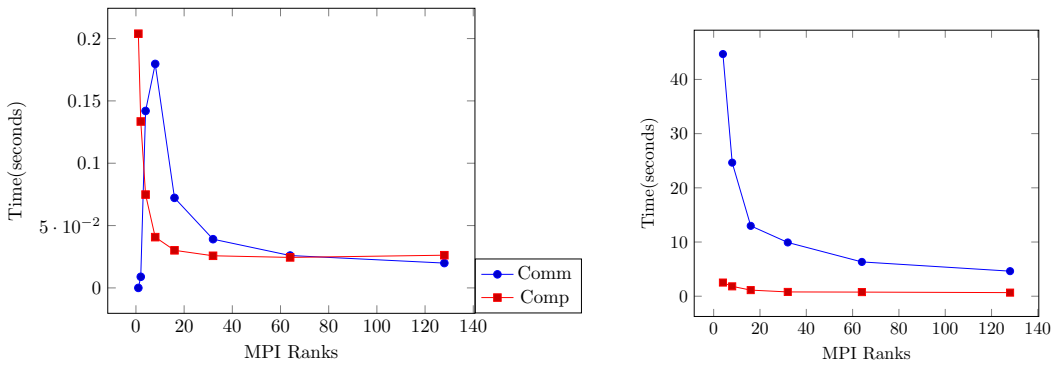


(a) PDE (b) Social Network

Fig. 5.2: Zoltan and D1 strong scaling on select (a) PDE and (b) Social Network graphs.

For graph processing in general, it is often difficult to demonstrate good strong scaling relative to single node runs. From the Graph500.org benchmark (June 2020 BFS results) [2], the relative per-node performance difference in the metric of "edges processed per second" between the fastest multi-node results and fastest single node results are well over 100x. For coloring on GPUs, graphs that can fit into a single GPU do not provide sufficient work parallelism for large numbers of GPUs, and multi-GPU execution incurs communication overheads and additional required rounds for speculative coloring. However, on average over all the graphs for which we have results, D1 still shows a 5.4x speedup over the single GPU run on 128 GPUs. On small or highly skewed graphs that fit on a single GPU, we do not see much speedup, due to the communication overheads and work imbalances that result from distribution even with relatively good partitioning.

On average over all our graphs, D1 sees a 47.2% increase in the number of colors from the single GPU run, while Zoltan sees an 53.6% increase in color use over the single GPU run. Such large color usage increases are mostly due to the Mycielskian19 and Mycielskian20 graphs. These graphs were generated to have known minimum number of colors (chromatic numbers) of 19 and 20 respectively, and our single GPU runs use 19 and 21 colors to color those graphs. Both our approach and the Zoltan implementation have trouble coloring these graphs in distributed memory, but our D1 implementation colors these graphs in fewer colors than Zoltan. Without these two outliers, the average color increase from the single GPU run is only 3.15% for D1, and Zoltan decreases color usage by 0.1% on average. Zoltan's smaller observed increase is due to its inherently lower concurrency giving a better quality coloring.



(a) Queen_4147 (b) com-Friendster

Fig. 5.3: D1 communication time (Comm) and computation time (Comp) from 1 to 128 GPUs.

Figure 5.3 shows the total communication and computation time associated with each run. For the Queen_4147 graph, computation time is the dominant factor in the larger rank runs. Figure 5.3(a) shows that we initially see computational scaling that levels off for large numbers of ranks. The computation time includes any computational imbalance and the time needed to launch GPU kernels. At high GPU counts, imbalance or kernel launches are likely the dominant component of the computation time for this graph, causing scaling to drop off above 64 GPUs. The com-Friendster graph shows computational scaling all the way to 128 GPUs, but, in this case, communication is the dominant factor of the execution time; computation scaling is not visible in the plot.

5.3. Distance-1 Weak Scaling. The greatest benefit of our approach is its ability to efficiently process massive-scale graphs. We demonstrate this benefit with a weak-scaling study conducted with uniform 3D hexahedral meshes. The meshes were partitioned with block partitioning along a single axis, resulting in the mesh being distributed in "slabs." Larger meshes were generated by doubling the number of elements in a single dimension to keep the per-process communication and computational workload constant. We run with up to 100 million vertices per GPU, yielding a graph of 12.8 billion vertices and 76.7 billion edges in our largest tests – **this graph was colored in less than half a second**.

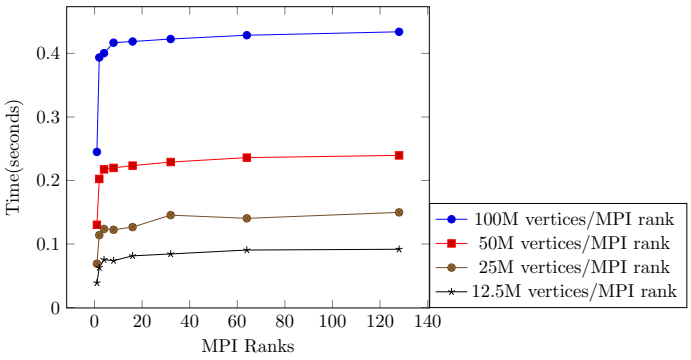


Fig. 5.4: Distance-1 weak scaling of D1 on 3D mesh graphs. Tests use 12.5, 25, 50, and 100 million vertices per GPU.

Figure 5.4 shows that the weak scaling behavior for D1 is very consistent. After a jump in execution time from one to two GPUs due to conflict resolution, the overall time increase from 2 to 128 GPUs is roughly 10% for each workload.

5.4. D1-2GL Performance. In general, D1-2GL does reduce the number of collective communications used in the distributed distance-1 coloring. Figure 5.5 compares the number of rounds for D1 and D1-2GL. Unfortunately, due to the increased cost of each communication round, D1-2GL does not generally achieve a speedup over D1. Additionally, second ghost layer vertices may be recolored if they are boundary vertices on another processor; this occurs often in dense inputs and incurs further communication costs. However, in distributed system with much higher latency, D1-2GL could be beneficial.

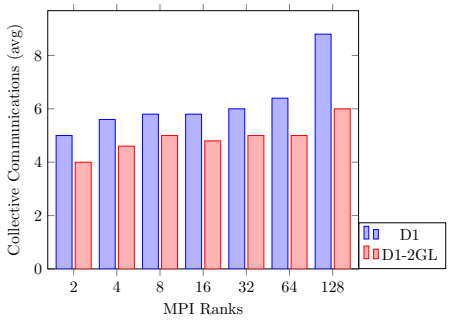
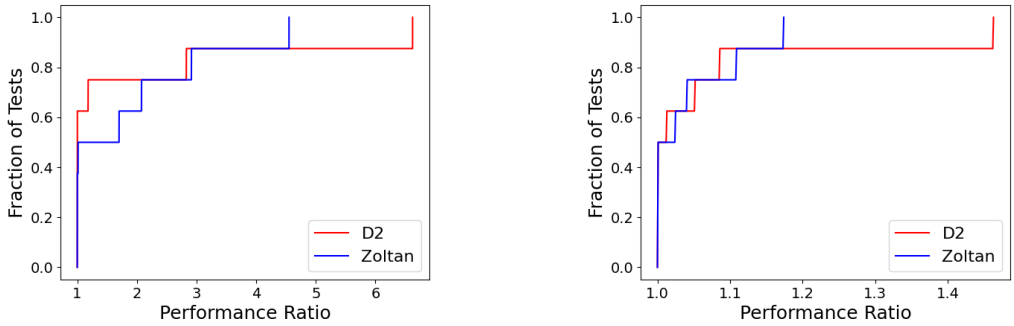


Fig. 5.5: Number of communication rounds for D1 and D1-2GL on Queen_4147 from 2 to 128 ranks.

5.5. Distance-2 Performance. We also compare our D2 method to Zoltan's distance-2 coloring using eight graphs from Table 3.1: Bump_2911, Queen_4147, hollywood-2009, europe_osm, rgg_n_2_24_s0, ldoor, Audikw_1, and soc-LiveJournal1. We use the same experimental setup as with the distance-1 performance comparison. Figure 5.6(a) shows that D2 compares well against Zoltan in terms of execution time, with D2 outperforming Zoltan on a majority of graphs. In the best case, we see a 4.5x speedup over Zoltan on the europe_osm graph.



(a) Runtime performance profile

(b) Color performance profile

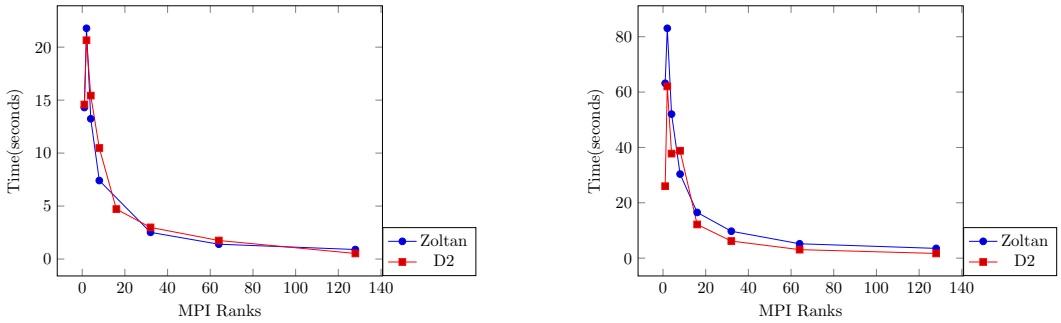
Fig. 5.6: Performance profiles comparing D2 on 128 Tesla V100 GPUs with Zoltan's distance-2 coloring on 128 Power9 cores in terms of (a) execution time and (b) number of colors computed for a subset of graphs listed in Table 3.1.

Figure 5.6(b) shows that D2 has similar color usage as Zoltan. D2 and Zoltan each produce the lower number of colors in half of the experiments. In all but one of the cases in which Zoltan uses fewer colors, D2 uses no more than 10% more colors. Interestingly, the number of colors used by D2 on the soc-LiveJournal1 graph is unchanged with one and 128 GPUs.

5.6. Distance-2 Strong Scaling. Figures 5.7(a) and 5.7(b) show the strong scaling behavior of D2 and Zoltan on Bump_2911 and Queen_4147. Bump_2911 shows similar scaling for both Zoltan and D2. For 128 ranks on Bump_2911, D2 uses 1.2% more colors than Zoltan, but runs 1.7x faster than Zoltan. With Queen_4147, D2 shows a brief scaling plateau from four to eight GPUs. This performance is an artifact of graph partitioning; the boundary size for eight ranks is the second largest out of all GPU counts except for 128 GPUs, resulting in a larger-than-expected communication cost. Zoltan is less sensitive to boundary sizes in the distance-2 case because it uses a more optimized communication pattern than our approach. After the eight-rank run, D2 scales slightly better than Zoltan up to 128 ranks. For the 128-rank run, D2 runs 2.1x faster than Zoltan, and uses 10% fewer colors.

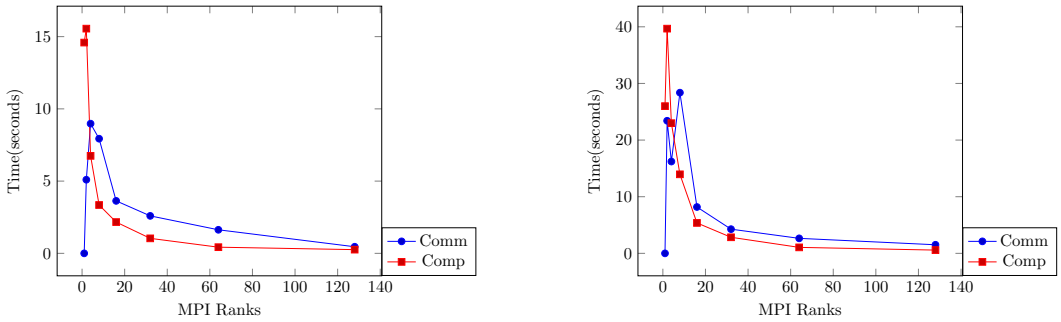
On average over the eight graphs, D2 exhibits 9.32x speedup on 128 GPUs over a single GPU, and uses 2.7% more colors than single GPU runs. Speedup is greater with D2 than D1 because distance-2 coloring is more computationally intensive, and thus has a larger work-to-overhead ratio.

Figures 5.8(a) and 5.8(b) show the communication and computation breakdown of D2 on Bump_2911 and Queen_4147. Bump_2911 shows computation and communication scaling for up to 128 ranks, while color usage increases by only 2.7%. In general, the relative increase



(a) Bump_2911 (b) Queen_4147

Fig. 5.7: D2 and Zoltan strong scaling for distance-2 coloring.



(a) Bump_2911 (b) Queen_4147

Fig. 5.8: D2 communication time (comm) and computation time (comp) from 1 to 128 GPUs.

in color usage from a single rank for distance-2 coloring is less than for distance-1 coloring. The number of colors used for distance-2 coloring is greater than for distance-1; therefore, a similar absolute increase in color count results in a lower proportional increase. Figure 5.8(b) shows that communication is the source of the runtime plateau shown in Figure 5.7(b).

5.7. Distance-2 Weak Scaling. Figure 5.9 demonstrates the weak scaling behavior for D2. The same hexahedral mesh graphs were used as in the D1 weak scaling experiments. For small per-node workloads, the weak scaling is good. For larger per-node workloads, execution times increase slightly. For these larger tests, the communication time, conflict detection time and initial coloring time all stay relatively flat, as does the number of rounds of communication. However, we observe an increase in imbalance for recoloring times across GPUs in these instances. Identifying and correcting the source of this imbalance is future work.

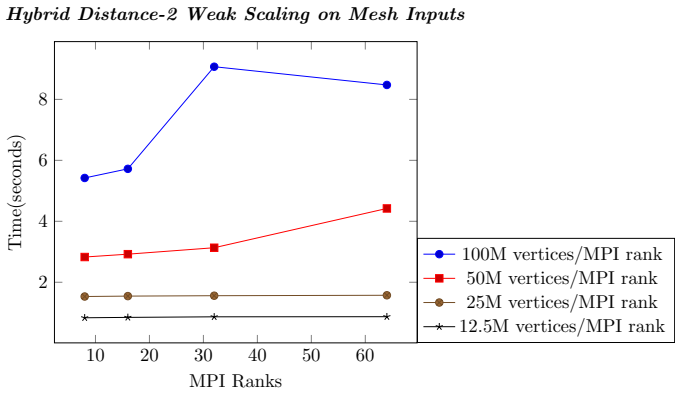


Fig. 5.9: Distance-2 weak scaling of D2 on 3D mesh graphs.

6. Future work. We plan to extend our distance-2 coloring to partial distance-2 coloring to support automatic differentiation applications. In partial distance-2 coloring, coloring criteria are applied only to vertices that are two hops apart. Since the colors of adjacent vertices are not considered, a proper partial distance-2 coloring may not be a proper distance-2 or even a proper distance-1 coloring. Our goal is to deliver a complete suite of MPI+X algorithms for distance-1, distance-2, and partial distance-2 coloring in the Zoltan2 package of Trilinos. This work's target application is the optimization of the computation of sparse Jacobian [25] and Hessian matrices [17], both of which are used in automatic differentiation and other computational problems [5].

7. Acknowledgments. We thank the Center of Computational Innovations at RPI for maintaining the equipment used in this research, including the AiMOS supercomputer supported by the National Science Foundation under Grant No. 1828083. This research was also supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525.

REFERENCES

[1] *Kokkos Kernels*, 2017.
[2] *Graph 500*, 2020.
[3] J. Allwright, R. Bordawekar, P. Coddington, K. Dincer, and C. Martin, *A comparison of parallel graph coloring algorithms*, SCCS-666, (1995), pp. 1–19.
[4] I. Bogle, E. G. Boman, K. Devine, S. Rajamanickam, and G. M. Slota, *Distributed memory graph coloring algorithms for multiple gpus*, To appear in Proc. of the 10th Workshop on Irregular Applications: Architectures and Algorithms at SC20, (2020).
[5] D. Bozdağ, Ü. V. Çatalyürek, A. H. Gebremedhin, F. Manne, E. G. Boman, and F. Özgüner, *Distributed-memory parallel algorithms for distance-2 coloring and related problems in derivative computation*, SIAM Journal on Scientific Computing, 32 (2010), pp. 2418–2446.
[6] D. Bozdağ, A. H. Gebremedhin, F. Manne, E. G. Boman, and U. V. Catalyurek, *A framework for scalable greedy coloring on distributed-memory parallel computers*, Journal of Parallel and Distributed Computing, 68 (2008), pp. 515–535.

[7] D. Brélaz, *New methods to color the vertices of a graph*, Communications of the ACM, 22 (1979), pp. 251–256.
[8] R. L. Brooks, *On colouring the nodes of a network*, in Mathematical Proceedings of the Cambridge Philosophical Society, vol. 37, Cambridge University Press, 1941, pp. 194–197.
[9] Ü. V. Çatalyürek, J. Feo, A. H. Gebremedhin, M. Halappanavar, and A. Pothen, *Graph coloring algorithms for multi-core and massively multithreaded architectures*, Parallel Computing, 38 (2012), pp. 576–594.
[10] T. A. Davis and Y. Hu, *The university of florida sparse matrix collection*, ACM Trans. Math. Softw., 38 (2011), pp. 1:1–1:25.
[11] M. Deveci, E. G. Boman, K. D. Devine, and S. Rajamanickam, *Parallel graph coloring for manycore architectures*, in 2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS), IEEE, 2016, pp. 892–901.
[12] K. D. Devine, E. G. Boman, L. A. Riesen, U. V. Catalyurek, and C. Chevalier, *Getting started with Zoltan: A short tutorial*, in Dagstuhl Seminar Proceedings, Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2009.
[13] H. C. Edwards, C. R. Trott, and D. Sunderland, *Kokkos: Enabling manycore performance portability through polymorphic memory access patterns*, Journal of Parallel and Distributed Computing, 74 (2014), pp. 3202–3216.
[14] M. Garey, D. Johnson, and H. So, *An application of graph coloring to printed circuit testing*, IEEE Transactions on Circuits and Systems, 23 (1976), pp. 591–599.
[15] A. H. Gebremedhin and F. Manne, *Scalable parallel graph coloring algorithms*, Concurrency: Practice and Experience, 12 (2000), pp. 1131–1146.
[16] A. H. Gebremedhin, D. Nguyen, M. M. A. Patwary, and A. Pothen, *Colpack: Software for graph coloring and related problems in scientific computing*, ACM Transactions on Mathematical Software (TOMS), 40 (2013), p. 1.
[17] A. H. Gebremedhin and A. Walther, *An introduction to algorithmic differentiation*, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 10 (2020), p. e1334.
[18] A. V. P. Grosset, P. Zhu, S. Liu, S. Venkatasubramanian, and M. Hall, *Evaluating graph coloring on GPUs*, ACM SIGPLAN Notices, 46 (2011), pp. 297–298.
[19] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, et al., *An overview of the Trilinos project*, ACM Transactions on Mathematical Software (TOMS), 31 (2005), pp. 397–423.
[20] M. F. Hoemmen, *Tpetra project overview.*, tech. rep., Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), 2015.
[21] M. T. Jones and P. E. Plassmann, *A parallel graph coloring heuristic*, SIAM Journal on Scientific Computing, 14 (1993), pp. 654–669.
[22] M. Naumov, P. Castonguay, and J. Cohen, *Parallel graph coloring with applications to the incomplete-lu factorization on the gpu*, tech. rep., NVidia White Paper, 2015.
[23] M. Osama, M. Truong, C. Yang, A. Buluç, and J. D. Owens, *Graph coloring on the GPU*, in GrAPL: Workshop on Graphs, Architectures, Programming, and Learning (IPDPSW), 2019.
[24] G. Rokos, G. Gorman, and P. H. Kelly, *A fast and scalable graph coloring algorithm for multi-core and many-core architectures*, in European Conference on Parallel Processing, Springer, 2015, pp. 414–425.
[25] M. A. Rostami and H. M. Bücker, *Preconditioning Jacobian systems by superimposing diagonal blocks*, in International Conference on Computational Science, Springer, 2020, pp. 101–115.
[26] A. E. Sariyüce, E. Saule, and Ü. V. Çatalyürek, *Scalable hybrid implementation of graph coloring using MPI and OpenMP*, in 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum, IEEE, 2012, pp. 1744–1753.
[27] G. M. Slota, S. Rajamanickam, K. Devine, and K. Madduri, *Partitioning trillion-edge graphs in minutes*, in 2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS), IEEE, 2017, pp. 646–655.
[28] G. M. Slota, S. Rajamanickam, and K. Madduri, *A case study of complex graph analysis in distributed memory: Implementation and optimization*, in International Parallel & Distributed Processing Symposium (IPDPS), 2016.
[29] M. K. Taş, K. Kaya, and E. Saule, *Greed is good: Optimistic algorithms for bipartite-graph partial coloring on multicore architectures*, arXiv preprint arXiv:1701.02628, (2017).

IMPROVEMENTS TO THE PERFORMANCE PORTABILITY OF BOUNDARY CONDITIONS IN ALBANY LAND ICE

MAX CARLSON[†], JERRY WATKINS[‡], AND IRINA TEZAUR[§]

Abstract. In order to accommodate the performance needs for the new generation of supercomputers, Sandia's Albany Land-Ice (ALI) code base is being refactored to use the Kokkos performance portability framework. One of the goals of this summer internship was to identify any performance bottlenecks related to the Kokkos refactor specific to the finite element assembly in ALI that would hinder GPU performance. Using the Land Ice Enthalpy problem as a test case, we identified performance bottlenecks related to the evaluation of the boundary conditions in Albany. By reworking how boundary conditions are represented in Albany, all memory accesses on the GPU are now properly coalesced, resulting in performant GPU kernels. Additionally, the new boundary condition layouts significantly reduce the amount of memory needed to store boundary data. The Kokkos refactor as a whole led to a finite element assembly speedup of 1.6× on 64 KNL CPU processors, and 4.5× on 64 V100 GPUs on a Greenland ice sheet problem discretized using a variable resolution 1km-10km mesh. With the reworking of boundary conditions, we achieved an additional 1.5× speedup on V100 GPUs. The overall performance improvements that come from this work are a large step towards the goal of full end-to-end ALI GPU runs on the Summit supercomputer.

1. Introduction. In an effort to modernize existing climate science software, there has been a strong push in recent years towards developing and upgrading codes to achieve high performance on the wide variety of specialized accelerator architectures available. As part of these modernization efforts, the Albany Land Ice (ALI) code is in the process of being updated to achieve high performance on both GPU and CPU architectures using the Kokkos [5] performance portability framework. In this paper, we describe some recent efforts towards a full refactor of ALI using Kokkos.

Albany [9] is an object-oriented, parallel, C++ code for discretizing and solving partial differential equations (PDEs) using the finite element method on unstructured grids. It is built to take full advantage of the Trilinos [7] collection of libraries and is based on a component-based design ideology that allows for quick and easy integration of new functionality. The typical problem-solving pipeline in Albany consists of constructing a global system corresponding to a discretized PDE, and passing it to Trilinos for a linear/nonlinear solve, or time-advancement (in the case of an unsteady problem).

As part of the initiative of modernizing climate science software, the ALI module in Albany was created to enable the simulation of ice sheet evolution using modern C++. ALI is based on the first-order approximation of the nonlinear Stokes flow model for glaciers and ice sheets (also known as the Blatter-Pattyn model), which models an ice sheet as a power-law viscous, incompressible fluid [10]. Toward this effect, the ALI model consists of a set of steady equations for the ice sheet velocities, which are coupled to dynamic equations for the ice thickness and ice temperature (or, equivalently, enthalpy). The ALI model is hooked up to the U.S. Department of Energy's (DOE's) Energy Exascale Earth System Model (E3SM) through its interface to the Model for Prediction Across Scales (MPAS) framework. The resulting code, known as MPAS-Albany Land Ice (MALI), enables dynamic simulations of ice sheet evolution, either in a stand-alone mode, or coupled to the E3SM. Attention is focussed herein on refactoring the finite element assembly (FEA) part of the so-called velocity and enthalpy solvers comprising MALI, both implemented on the ALI side of the MALI code. The FEA in Albany in general takes approximately 50% of the total CPU time when running an ALI problem. The remaining 50% of the CPU time is spent in the linear solve part of the computation, the discussion of which is beyond the scope of this paper.

[†]University of Utah School of Computing, mcarlson@cs.utah.edu
[‡]Sandia National Laboratories, jwatkin@sandia.gov
[§]Sandia National Laboratories, ikalash@sandia.gov

One of the biggest challenges for modern scientific software development is achieving high performance on the various multicore and manycore architectures available. Each architecture can have its own unique programming model, API, and specific requirements to achieve high performance. Previous work has been done to study and refactor the Albany code using the Kokkos performance portability framework and the work presented in this paper is the next step in this process [6, 11]. This work focused on achieving a performance portable implementation of the FEA for the ALI velocity problem, not including boundary condition effects. Herein, we describe our recent work in porting to Kokkos the FEA assembly for the ALI enthalpy problem [3, 8], which, when coupled to the ALI velocity problem, enables a more complete characterization of an ice sheet's state, including its velocity together with its temperature. We discuss also our resolution of several issues that originally prevented the high performance evaluation of the boundary conditions for the ALI enthalpy problem.

The remainder of this paper is structured as follows. First, in Section 2, we give some background about performance portability and describe some of the performance considerations that are unique to GPU architectures. Next, in Section 3, the specific implementation details for the enthalpy and boundary condition performance refactor is presented. Finally, in Section 4, we present the performance we achieved using the techniques described in Section 3 on several different architectures.

2. Performance Portability. While bridging the gap between all of the programming models and accelerator architectures is not a trivial or solved problem, a number of frameworks have been developed to tackle this issue. Some of the frameworks that have been designed for this task include Kokkos [5], Raja [4], OneAPI [2], HIP [1], and others. Many of these solutions overlap in terms of functionality and the choice of framework largely boils down to taste and project requirements. Since the Trilinos project has chosen to use Kokkos to achieve performance portability and since Albany is so tightly coupled to Trilinos, it made sense to use the Kokkos framework for this work.

The main target for the work outlined in this paper is to achieve high performance on GPUs to enable Albany Land Ice runs that take full advantage of the Summit supercomputer's hardware configuration. Achieving high performance on Intel's Knight's Landing (KNL) generation of CPUs is a secondary target. Attaining high performance on GPUs is especially tricky due to performance needs that are unique to GPU architecture (described in more detail below). Since much of this work requires a familiarity with these unique needs, we will now give a little background on concepts that will be referenced throughout the remainder of the paper.

2.1. Unified Virtual Memory. Unlike traditional CPU architecture, GPUs can only access data that are stored on the device itself and have a unique memory hierarchy. The cost of moving data between the host and the device is significant and care must be taken to reduce these data movements as much as possible. Typically, host/device data transfers are controlled explicitly by the programmer. While this allows for a greater amount of control, the added code complexity and programmer effort can sometimes not be worth the cost.

Alternatively, a project can enable CUDA's unified virtual memory (UVM) mode to avoid handling these memory transfers explicitly. UVM is an abstraction that treats the host and device as a single memory space and does device transfers as needed behind the scenes whenever memory is accessed. While this mode makes handling data movement much simpler for the programmer, it may cause performance degradations, as device transfers may occur in unexpected places.

Since Trilinos is designed to use UVM, Albany has inherited this requirement. We point this out because UVM has non-intuitive effects on performance measurements. For instance,

the first time an evaluator accesses data that was initialized on host, there will be a hidden device transfer that gets captured in the evaluator timing. There are known workarounds to avoid using UVM in conjunction with Trilinos, but the developers of Trilinos are in the process of removing UVM as a requirement. Therefore, we decided it was not worth the effort and increased code complexity to utilize such a workaround for this project.

2.2. Maximizing Available Parallelism. Achieving high GPU performance requires exposing enough parallelism to fully take advantage of available computing hardware on the GPU. If the amount of work to be done across the threads is not large enough, many of the device's streaming multiprocessors (SM) will sit idly while waiting for the result. It is sometimes unavoidable that a stage of computation may not have enough work to fully saturate the device. In this case, it is a good idea to schedule multiple small kernels concurrently so that the device can be fully utilized. While scheduling concurrent kernels can be done manually using CUDA streams, newer versions of CUDA have introduced the CUDA graph structure. A CUDA graph allows a programmer to provide a directed acyclic graph of kernels to be launched and their dependencies can be used by CUDA to attempt to maximize concurrent scheduling of computation. Additionally, work is being done by the Kokkos team to incorporate CUDA graphs into the Kokkos library.

2.3. Coalesced Memory Access. When the GPU is saturated with work, coalesced memory access becomes the most important aspect of achieving high performance. A GPU block loads in data as a contiguous block that can then be processed by its collection of threads. If each thread requires a piece of data from different locations in memory, then the SM will have to load an entire block's worth of data for each thread. If, alternatively, all of the threads require data that are contiguous in memory, the SM will only have to fetch one single block. This type of contiguous memory access is referred to as a coalesced memory access. GPUs are sometimes capable of enormous speedups over CPUs but if a programmer does not ensure coalesced access then it is possible to achieve slowdowns over CPU of equally large magnitude.

3. Implementation Details. Albany's finite element assembly relies on the Trilinos Phalanx package that decomposes a complex problem into smaller pieces known as evaluators with managed dependencies. These evaluators then form a directed acyclic graph of operations to be carried out. In general, an evaluator is used to calculate both a residual and a corresponding Jacobian. Jacobians are computed using the automatic differentiation package within Trilinos known as Sacado.

In this section, we will split the details of the changes we made to the enthalpy problem into those that only affected the volume evaluators and those that only affected the boundary condition evaluation. Volume evaluators are those that evaluate the residual and Jacobian across the entire computational domain (volume). We separate these two works since the volume refactor can be seen as a simple extension of the work done in [6] to the enthalpy problem. The boundary condition refactor is more involved and required fundamentally rethinking how boundary conditions are represented by Albany in order to ensure coalesced memory access for consistently achieving high performance on GPU architectures.

The typical pipeline for finite element assembly in Albany is shown in Figure 3.1. Problem data are loaded during the import stage and distributed via MPI communication to data structures that allow for efficient shared memory access during the interpolation and evaluation phase. Once these phases complete, the resulting computed fields are scattered using MPI communication and exported to Trilinos to be used in the solver. For the enthalpy problem, the phases that still needed to be ported using Kokkos reside entirely in the interpolation and evaluation phase of this pipeline.

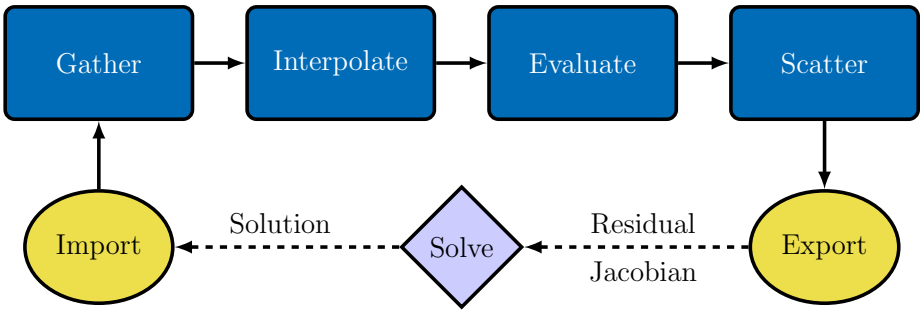


Fig. 3.1: A flow chart showing the main work flow for a single nonlinear iteration of the Albany Land Ice solver. Finite element assembly begins and ends with distributed memory assembly (DMAssembly) (in yellow) which constructs data structures which run more efficiently during the linear solver and finite element assembly phases. This is also where MPI communication occurs. The remaining shared memory assembly (SMAssembly) processes (in blue) perform global and local assembly and are parallelized over elements using Kokkos parallel_for.

3.1. Volume Refactor. Assembly of the enthalpy problem's system is done by running a series of evaluators, each with dependent and evaluated fields. Some of these evaluators fall under the category of 'Interpolate' such as interpolating a field from cell centers to quadrature points or nodal values. The 'Evaluate' phase consists of using interpolated values to compute fields relating to the science of the underlying problem. For the enthalpy problem this includes evaluating the hydrostatic pressure or temperature throughout the domain. Since the evaluators in the 'Interpolate' phase are common among other problems in Albany, those that are specific to volume evaluation have already been ported. This section then focuses on the physics evaluation in the 'Evaluate' phase.

Each evaluator in the 'Evaluate' phase has a common structure. Each evaluator loops at the highest level over the total number of cells in the workset. Then depending on the type of evaluator, there may be a loop over quadrature points, cell nodes, or vector dimensions. At the cell level, each computation is entirely data independent and makes up the majority of the computation. Therefore, our strategy is to parallelize this loop using a Kokkos 'parallel_for'. Since the fields being evaluated are represented as Kokkos views, the underlying data layout is chosen depending on the hardware. For GPUs, the default is known as "Kokkos LayoutLeft". This layout is such that the first index of a view, in this case cells, is the contiguous one and ensures that the memory access is coalesced.

For simpler evaluators, simply parallelizing over the total number of cells is enough to achieve high performance. However, to compute the enthalpy residual, the evaluator has multiple stages of computation that each has its own cell loop. First, the evaluator loops over all cells to compute an optional dissipation term. Then, another loop over the cells to compute the influence of the basal term. And then finally, a third loop over the cells to compute the total residual. This was inefficient since each of these loops update every entry of the enthalpy residual field. By combining these into a single loop over cells, and accumulating the enthalpy residual into a local intermediate value, we instead only write to the enthalpy residual field once at the very end of computation. Since accessing global memory on GPUs is slow, reducing the number of times the evaluator needs to move data to global memory is ideal. While this data movement has less of an impact on CPUs, it does reduce the number of writes, potentially leading to improved performance.

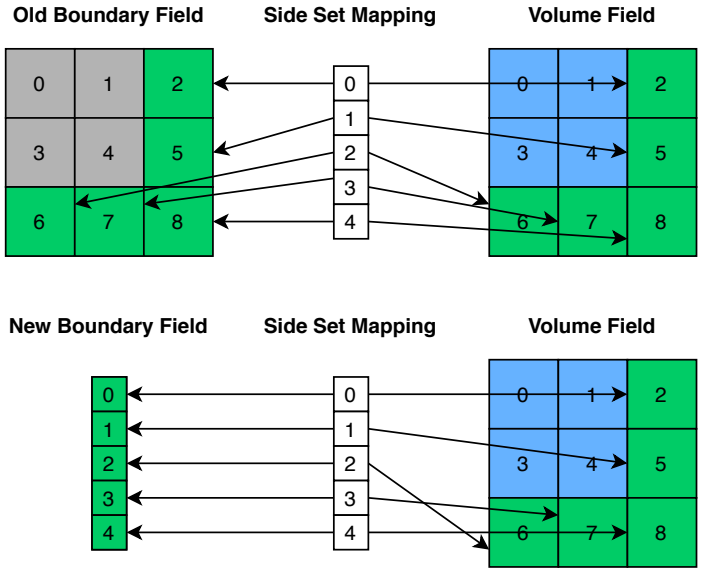


Fig. 3.2: Diagram illustrating the old and new layouts for boundary data fields. Grey blocks correspond to cells in the boundary field that will never be accesses at any point. Green cells correspond to entries corresponding to the boundary being represented. Blue cells are cells in the volume field that will be accessed at some point but are not necessarily part of a boundary computation. Notice that the new boundary layout matches the side set mapping structure so both can be accessed in coaleseced manner on the GPU.

3.2. Boundary Condition Refactor. Before this refactor, the fields corresponding to boundary conditions were represented by arrays over all of the cells in the domain including those that are not related to the boundary at all. Then, instead of looping over the total cells, a boundary condition evaluator would loop over an array of side set structures that would point to the cells relevant to the boundary condition calculation. With this representation, only a very small percentage of the underlying field contained data that would ever be accessed or updated.

The obvious strategy for parallelizing these evaluators was to parallelize over the array of side set structures. This way, accessing the array of side set structures would be coalesced and each element of the side set array would be data independent. However, each side set structure only contained the information of where to find the cell/side pair of interest in the underlying field data structure. Each entry in the side set array was contiguous, but the locations that they pointed to were not. This meant that the bulk of computation required accessing global memory in an entirely non-coalesced fashion, which resulted in very large performance penalties.

In order to get around this issue, the underlying data layout for boundary fields had to be modified and a small visual example of the new layout can be seen in Figure 3.2. Instead of having boundary fields scattered throughout an array with many unused entries, the updated layout was collapsed to match that of the side set structure array. In this case, the i^{th} entry of the side set array corresponded to the i^{th} entry of the field array. With this structure, both access to the field of interest and the side set array could be done in a coalesced manner on the GPU. The side set information was still required to eventually

gather the boundary data into volume fields, this non-coalesced access would only happen once instead of every single time boundary data were accessed.

In addition to updating the layouts, the side set structure itself was not in a form that was accessible by GPU. Originally, side sets were represented by a C++ vector of structures containing side set index information. In order to make this accessible by the GPU, we had to replace this structure with a Kokkos view for each item in the original side set structure. Also, since Albany separates work into worksets and boundary data is defined only for each workset, the structure-of-views were separated into a global and local structure. The global structure contained the entirety of the side set information across all worksets and the local structure pointed to specific subviews limited to the workset in question.

4. Performance Results. In order to test the performance of the refactored code against the original, we set up a collection of experiments with three different types of builds for each architecture. These three builds are: the original code, the volume refactor code, and the combined volume and boundary condition code (BCs). By doing this, we were able to determine exactly how much each refactor contributed to the overall performance change.

Since we are using Kokkos to handle the performance portability aspects, we can also provide architecture information during compilation to target specific architectures. The three types of builds are 'Serial', 'OpenMP', and 'GPU'. By providing Kokkos with this information, the underlying data layouts can then be defined at compile time using a template parameter.

The performance results in this section were all computed on either the Vortex or Mutrino computing clusters at Sandia. Vortex is a cluster that has 4 V100 GPUs per node and a dual-socket POWER9 CPU (40 cores per node). For OpenMP runs, we used Mutrino which has single-socket Intel Knight's Landing (68 cores per node) and dual-socket Haswell generation CPU nodes (32 cores per node).

For these comparisons, we ran the enthalpy problem on a Greenland ice sheet geometry discretized using a variable resolution 1km-10km mesh. To solve this problem on GPUs, we used 64 Volta V100 GPUs distributed among 16 nodes on Vortex. For OpenMP runs, we used a comparable setup of 16 single-socket KNL CPUs (68 cores per socket) distributed among 16 nodes on Mutrino using 4 OpenMP threads per core mapped to hardware threads. The first experiment we ran was to see how each build compares with the original non-refactored implementation and the results can be seen in Tables 4.3, and 4.4.

The temperature, dissipation, hydrostatic pressure, and liquid water fraction evaluators all have the same basic internal structure. They are simply a loop over the number of cells and quadrature points to update a single independent value. Even though each of these evaluators operate on the same volume of data, timing each evaluator individually shows that each takes a drastically different amount of time. This is a timing artifact that occurs while using CUDA UVM. When using CUDA UVM, data transfers are only done as soon as host-side data are first requested. Hydrostatic pressure, and liquid water fraction are some of the first evaluators to be called and their timings are artificially inflated by data transfers. From this experiment, it is clear that we cannot rely on timing individual evaluators to determine the overall performance change. Therefore, we instead use Nvidia's profiling tool, nvprof, to evaluate a number of metrics for each of these individual evaluators.

The results of this profiling for the volume evaluator refactor can be seen in Table 4.1. The profiling results for the boundary condition refactor can then be seen in Table 4.2. In the volume evaluators, we see that on 8 total V100 GPUs, we are fully saturating the device and therefore getting device memory throughput close to the maximum attainable for V100s. Since the boundary condition evaluators have less data to operate on, even for

Table 4.1: Nvidia profiler (nvprof) results for volume refactor evaluators, run using two nodes with four V100 GPUs each on the 1km-10km variable resolution Greenland ice sheet mesh. Using this configuration, the volume evaluators have more than enough work to saturate the GPU and achieves high device memory throughput. Occupancy is largely near 100% but can be improved by limiting register usage.

Evaluator	**Achieved Occupancy**	**Device Memory Read Throughput**	**Device Memory Write Throughput**	**Global Load Efficiency**	**Global Store Efficiency**
Temperature	53.3%	371.8 GB/s	355.2 GB/s	100%	100%
Dissipation	96.2%	481.2 GB/s	235.3 GB/s	100%	100%
HydrostaticPressure	96%	509.7 GB/s	256.5 GB/s	100%	100%
LiquidWaterFraction	95.4%	512.8 GB/s	255.3 GB/s	100%	100%
EnthalpyResid	43%	571.5 GB/s	94.5 GB/s	100%	100%

Table 4.2: Nvidia profiler (nvprof) results for boundary condition refactor evaluators, run using two nodes with four V100 GPUs each on the 1km-10km variable resolution Greenland ice sheet mesh. The volume of data for these evaluators using this node configuration is enough to achieve mostly good throughput but they are becoming latency bound. These evaluators are a prime candidate for concurrent kernel launches to ensure full device utilization.

Evaluator	**Achieved Occupancy**	**Device Memory Read Throughput**	**Device Memory Write Throughput**	**Global Load Efficiency**	**Global Store Efficiency**
EnthalpyBasalResid	46.6%	364.4 GB/s	91.5 GB/s	88.3%	25%
BasalMeltRate	46.8%	463.9 GB/s	145 GB/s	100%	100%
InterpolationSide	46.8%	417.3 GB/s	167.2 GB/s	100%	100%
VecInterpolationSide	44.2%	468.7 GB/s	151.4 GB/s	100%	100%

Table 4.3: Total fill time comparison of MPI+CUDA V100 build for the original code (CUDA-original), the volume refactor (CUDA-volume), the combined volume and boundary condition refactor (CUDA-BCs), and MPI-only serial POWER9 build for the original code (serial-original).

	CUDA-original	CUDA-volume	Speedup
Total Fill Time	183.12 sec	3.8 sec	48.2
	serial-original	CUDA-volume	Speedup
Total Fill Time	16.75 sec	3.8 sec	4.4
	serial-original	CUDA-BCs	Speedup
Total Fill Time	16.75 sec	2.79 sec	6

Table 4.4: Total fill time comparison of MPI+OpenMP KNL build for the original code (KNL-original), the volume refactor (KNL-volume), and MPI-only serial KNL build for the original code (serial-original).

	KNL-original	KNL-volume	Speedup
Total Fill Time	28.12 sec	20.14 sec	1.4
	serial-original	KNL-volume	Speedup
Total Fill Time	32.4 sec	20.14 sec	1.6

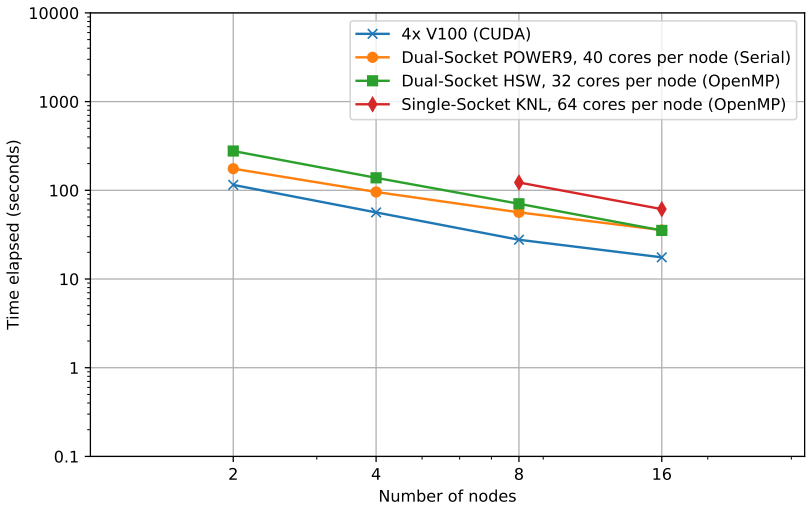


Fig. 4.1: Strong scaling for the Greenland ice sheet 1km-10km mesh using the fully refactored code.

this node configuration, we see that the device throughput is diminished in comparison. The boundary evaluators become latency bound much more quickly than the volume evaluators and are prime candidates for concurrent kernel launching using CUDA or Kokkos graphs.

Finally, we observe the overall total finite element assembly time for each build to see how the performance changes. In Table 4.3, we show the total fill time for the original CUDA build, the volume refactored CUDA build, and the original serial build. Since the original CUDA build was only partially ported to run on GPUs, every time a CPU evaluator followed a GPU evaluator (or vice versa), there was an expensive data transfer. This is partially why the original CUDA build has an artificially inflated run time. Additionally, we only launch one MPI task per GPU so the host-side work is divided only among a few processes. However, when compared to the MPI-only serial build, we see a greater than four times speedup.

While this project was primarily targeting GPU performance, we also wanted to see how the volume refactor affected performance on KNL CPUs. For this, we used the KNL OpenMP build and ran the same comparison. The results of this comparison can be seen in Table 4.4. The reader can observe that even without explicitly targeting OpenMP, the

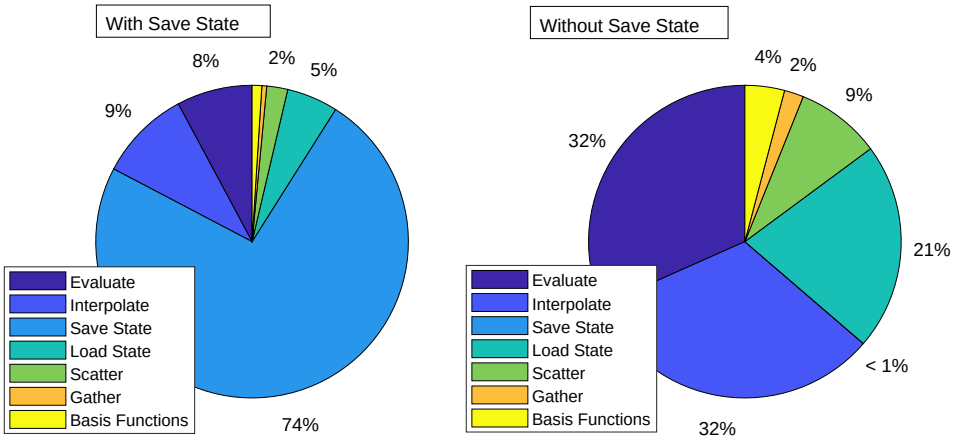


Fig. 4.2: Timing breakdown for all evaluator groups within the enthalpy finite element assembly using the fully refactored MPI+CUDA build on 64 V100 GPUs.

volume refactor attained a 1.4× speedup over the original OpenMP build and a 1.6× speedup over the MPI-only serial build.

We then added in the boundary condition refactor and repeated these comparisons to determine the performance change. In Table 4.3, we show that the boundary condition refactor increased the speedup over the original MPI-only serial build to 6×. By comparing the volume CUDA build and the fully refactored build, we see that the boundary condition changes give us an extra 1.4× speedup. We repeated this comparison for an OpenMP on a KNL machine, but there was no measurable change. This is to be expected since the boundary condition changes were purely targeted at GPUs.

We then ran a strong scaling experiment for the fully refactored code for each of the architectures using the Kokkos backend that achieved the best performance. In Figure 4.1, we present the total fill times using 2, 4, 8, and 16 nodes. At 16 nodes, the performance of the CUDA build starts to degrade since the amount of work per GPU is small and not enough to saturate the devices. This is further evidence that as the amount of work decreases per rank, we need to properly schedule concurrent launching of evaluators.

From the strong scaling experiment, we see that the CUDA build is only about 2.4 times faster than the Haswell build. In order to understand why the GPU build was not achieving the kind of speedup we expected, we ran a timing breakdown to determine what parts of the finite element assembly were taking the most time. In Figure 4.2, we show this breakdown with the evaluators grouped into categories. The 'Evaluate' category contains all of the physics based evaluators and contains the majority of evaluators that were refactored in the volume refactor. 'Interpolate' contains interpolation evaluators and, aside from the boundary interpolation evaluators, was already ported to GPU since they are used in other problems. 'Save State' and 'Load State' consists of evaluators that prepare the underlying data fields for I/O operations. 'Scatter' and 'Gather' correspond to the stages seen in Figure 3.1 and are detailed in [9]. Finally, 'Compute Basis Functions' contains evaluators the evaluate the finite element basis functions throughout the volume and boundary.

From these results, it is very clear that the performance bottleneck is the 'Save State' evaluators. These evaluators are needed in preparation for I/O operations which currently can not be accelerated using GPUs. We have a potential change in mind to the underlying mesh discretization that would remove the need for these evaluators but that is outside of

the scope of this paper. However, the performance penalty we are seeing here is only seen on the GPU build, so once these evaluators are removed, the overall speedup over Haswell should become much more pronounced.

Since the 'Save State' evaluators dwarf all of the other categories, we removed that category from the breakdown and the remaining breakdown can be seen in Figure 4.2. From this, it can be seen that the 'Evaluate' category is now roughly identical in total run time as the 'Interpolate' category which was previously ported to Kokkos. Aside from further optimization to these evaluators, we can now see that the port of the 'Evaluate' category has achieved the kind of performance we were aiming for.

5. Conclusion and Future Work. With the work outlined in this paper, the finite element assembly of the enthalpy problem is now running entirely on the GPU and attaining high performance relative to other architectures. Additionally, since we have used Kokkos for the GPU refactor, we are also seeing speedups on serial and KNL OpenMP builds for the enthalpy problem.

The work we did refactoring the enthalpy problem's boundary condition evaluation also acts as a proof of concept that we can attain good GPU performance for boundary evaluators. This is a big step forward since none of the boundary condition evaluators for other problem types implemented in Albany have been ported. Now that we can move boundary condition evaluation to the device without suffering performance penalties, we are one step closer to full end-to-end runs on the GPU clusters such as Summit.

Since the goal of this project was primarily to get the enthalpy problem running entirely on GPU, some of the evaluators are not as optimized as they could be. The next step in this project is to revisit both the volume and boundary evaluators and optimize as needed to make sure we are getting as much performance as possible. We also intend to use the same boundary condition implementation to refactor the boundary condition evaluation for the velocity problem. Finally, the remaining major bottleneck is the linear solve. Once we have finished porting the linear solve to GPU, we can then run end-to-end land ice simulations entirely on GPUs. Porting the linear solver also relies on porting of Trilinos code which is currently a work in progress.

While the dominating time for the full problem is now the linear solve, we can still achieve better scaling for the finite element assembly by taking advantage of concurrent scheduling of kernels on the device. As we decrease the problem size per GPU via strong scaling, it is clear that we are not achieving full utilization of the device. We can improve this scalability by giving the underlying Phalanx DAG to Kokkos and take advantage of CUDA/Kokkos graphs to attain a higher level of concurrency.

Acknowledgements. Support for this work was provided through the Scientific Discovery through Advanced Computing (SciDAC) program funded by the US Department of Energy (DOE), Office of Science, Advanced Scientific Computing Research and Biological and Environmental Research Programs.

REFERENCES

[1] *HIP API Documentation.* https://rocmdocs.amd.com/en/latest/Installation_Guide/HIP.html. [Online; accessed 27-August-2020].

[2] *Intel® oneAPI Programming Guide (Beta).* https://software.intel.com/content/www/us/en/develop/documentation/oneapi-programming-guide/top.html. [Online; accessed 27-August-2020].

[3] A. Barone and M. Perego, *Implementation of enthalpy model for polythermal glaciers*, CCR Summer Proceedings, (2016).

[4] Beckingsale, David and Hornung, Richard and Scogland, Tom and Vargas, Arturo, *Performance Portable C++ Programming with RAJA*, in Proceedings of the 24th Symposium on Principles and Practice of Parallel Programming, PPoPP '19, New York, NY, USA, 2019, Association for Computing Machinery, p. 455–456.

[5] Carter Edwards, H. and Trott, Christian R. and Sunderland, Daniel, *Kokkos*, J. Parallel Distrib. Comput., 74 (2014), p. 3202–3216.

[6] I. Demeshko, J. Watkins, I. K. Tezaur, O. Guba, W. F. Spotz, A. G. Salinger, R. P. Pawlowski, and M. A. Heroux, *Towards performance-portability of the Albany finite element analysis code using the Kokkos library*, The International Journal of High Performance Computing Applications, 33 (2019), pp. 332–352.

[7] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, et al., *An overview of the Trilinos project*, ACM Transactions on Mathematical Software (TOMS), 31 (2005), pp. 397–423.

[8] Hoffman, Matthew and Perego, Mauro and Price, Stephen and Lipscomb, W. and Jacobsen, Douglas and Tezaur, Irina and Salinger, Andrew and Tuminaro, Raymond and Zhang, Tong, *MPAS-Albany Land Ice (MALI): A variable resolution ice sheet model for Earth system modeling using Voronoi grids*, Geoscientific Model Development Discussions, (2018), pp. 1–47.

[9] A. G. Salinger, R. A. Bartlett, A. M. Bradley, Q. Chen, I. P. Demeshko, X. Gao, G. A. Hansen, A. Mota, R. P. Muller, E. Nielsen, et al., *Albany: Using component-based design to develop a flexible, generic multiphysics analysis code*, International Journal for Multiscale Computational Engineering, 14 (2016).

[10] I. K. Tezaur, M. Perego, A. G. Salinger, R. S. Tuminaro, and S. F. Price, *Albany/FELIX: A Parallel, Scalable and Robust Finite Element Higher-Order Stokes Ice Sheet Solver Built for Advanced Analysis*, Geoscientific Model Development, 8 (2015), pp. 1197–1220.

[11] J. Watkins, I. Tezaur, and I. Demeshko, *A study on the performance portability of the finite element assembly process within the Albany land ice solver*, 02 2020.

GENTEN PERFORMANCE PORTABLE DENSE TTM KERNELS

BENJAMIN COBB*, ERIC PHIPPS†, HEMANTH KOLLA‡, AND ÜMIT V. ÇATALYÜREK§

Abstract. This paper details several implementations of optimized, performance portable Tensor Times Matrix kernels utilizing the Kokkos programming model. The Tensor Times Matrix (TTM) kernel consists of multiplying a tensor by a matrix along a given mode of the tensor. The TTM kernel is used in several tensor algorithms, the most prominent of which is the Sequentially Truncated Higher-Order Singular Value Decomposition (ST-HOSVD). The ST-HOSVD can be used to facilitate data-analysis and compression of high dimensional data, such as combustion simulation data, which is currently an active Sandian research topic. Speeding up the ST-HOSVD algorithm heavily relies upon optimizing the TTM kernel, due to the fact that a significant, often majority, of the time in the ST-HOSVD algorithm is spent calculating the TTM at each iteration. Our TTM implementations are based on the work of Ballard et al. and are competitive with state of the art TTM implementations, with the added benefit of leveraging the Kokkos programming model to enable performance portability. We present comparison results with several of these implementations, discussing the advantages and disadvantages of each.

1. Introduction. As our ability for collecting and generating data continues to evolve, the corresponding capacity to store and quickly analyze such data lags behind. Colloquially known as the "storage bottleneck" problem, this phenomena is often seen in fields such as bioinformatics that rely on collecting, analyzing and storing vast amounts of data [13]. Furthermore, many modern simulations can also easily generate terabytes of data, as seen in the turbulent flame combustion simulations done by Kolla et al. [9]. In both instances the problem is exacerbated when the data is a higher-dimensional tensor as the size of the dataset grows exponentially with an increase in the number of dimensions. Many techniques have been developed to help compress such massive data in order to make it more amenable to storage and analysis. One of these techniques, similar to the Tucker decomposition, can be used to calculate the higher dimensional analog of the dataset's Singular Value Decomposition (SVD). Generally a low rank approximation of this decomposition is used in practice. This low rank higher order singular value decomposition is commonly referred to as the Sequentially Truncated Higher Order SVD, ST-HOSVD for short. The algorithm to calculate the ST-HOSVD primarily relies upon two tensor kernels: the Gram matrix referred to as the Gram kernel and the tensor times matrix kernel, referred to from here out as the TTM kernel. In this paper we present several implementations of the dense TTM kernel based on the work of Ballard et al. [2] that are competitive in terms of single-node runtimes with the state of the art. In addition, several of these kernels are implemented using the versatile Kokkos programming model, giving them the performance portability to run on heterogeneous systems and improving their chances of maintaining performance on future computer architectures. To our knowledge these implementations are the first implementation to apply Kokkos to the TTM kernel in this manner.

2. Related Work.

2.1. Kokkos. Kokkos [4] is a C++ programming model developed by Sandia National Labs that aims to enable *performance portable* code that is capable of running with comparable performance across a variety of manycore computer architectures. It does so through utilizing multi-dimensional array data structures with polymorphic layouts known as *Views* to manage architecture dependent data memory distributions. These *Views* have two primary memory layouts known as *LayoutLeft* and *LayoutRight*. LayoutLeft stores the View in

*Georgia Institute of Technology, bcobb33@gatech.edu
†Sandia National Laboratories, etphipp@sandia.gov
‡Sandia National Laboratories, hnkolla@sandia.gov
§Georgia Institute of Technology, umit@gatech.edu

column-major order, whilst LayoutRight stores the View in row-major order. Under normal circumstances Views keep a reference count that tracks the number of Views accessing the same memory location, automatically deallocating the memory once the last View referencing it is destroyed or goes out of scope. However, Views may also be *Unmanaged*, meaning that they do not increase the reference count for a View memory location. Unmanaged Views are essentially pointers to View memory and are thus useful for accessing View memory independent of preexisting Layout. The way in which these Views are accessed is specified by various Kokkos *execution spaces*, which reflects the device that is accessing the memory. For example, a *host execution space* may refer to CPU data accesses, whilst a *CUDA execution space* may refer to GPU data accesses. Kokkos additionally provides the ability to copy memory between these various execution spaces. Several of our TTM implementations utilize this Kokkos programming model to facilitate performance portability.

2.2. KokkosKernels. Kokkos Kernels is built on top of Kokkos to deliver performance portable math kernels for linear algebra operations and graph computations. We utilize Kokkos Kernel's general matrix-matrix multiplication (GEMM) function in several of our implementations to facilitate the tensor submatrix multiplications that are a key step in the TTM algorithm and are detailed in subsequent sections.

2.3. GenTen. GenTen [14] is the driving force behind this project as we seek to integrate our optimized TTM implementations into GenTen's ST-HOSVD function to yield the first ever full Kokkos implementation of this critical algorithm. GenTen currently contains a performance portable Matricized Tensor Times Khatri-Rao Product (MTTKRP) kernel that leverages Kokkos. GenTen uses this highly optimized MTTKRP kernel to facilitate its implementation of the Candecomp/Parafac Alternating Least Squares (CP-ALS) algorithm, another tensor decomposition technique that can be viewed as the higher-dimensional analog of matrix factorization. GenTen's MTTKRP implementation has been benchmarked extensively on modern Intel CPUs, Knight's Landing (KNL) accelerators, and NVIDIA GPUs. On this diverse set of architectures, GenTen's CP-ALS function performed with comparable efficiency on all systems. Future work includes testing our TTM implementations on a similar set of architectures in hopes of showing similar performance across architectures.

2.4. TuckerMPI. TuckerMPI [2] is a C++ software package that utilizes MPI to implement several tensor decomposition algorithms, including the aforementioned ST-HOSVD algorithm. TuckerMPI's goal is to provide a framework for parallelizing massive tensor computations across multiple CPU-based nodes. On each node, the local computations can be parallelized via multiple on-node MPI processes, each of which utilizes highly optimized LAPACK libraries for their individual computations. TuckerMPI provides the user with the ability to set the dimensions of the processor grid upon which the tensor is block partitioned. TuckerMPI has been shown to effectively compute the ST-HOSVD of datasets up to 6.7 TB, resulting in compression ratios up to 4×10^4. Originally we had extracted TuckerMPI's TTM from its ST-HOSVD driver and compared its runtime to GenTen's runtime. However, due to high variance in the collected timings we ultimately decided to not make the comparison. Our TTM implementations were closely based off of the algorithm specified in Ballard et al [2].

2.5. Matlab Tensor Toolbox. Within the last decade, the Matlab Tensor Toolbox (TTB) [8] has arguably been the most seminal of all tensor software packages in the field of tensor decompositions. The TTB allows for fast prototyping and experimentation of tensor computations. TTB's TTM function relies on Matlab's highly tuned LAPACK libraries and builtin thread parallelism to efficiently execute tensor calculations on a single node. TTB's dense TTM kernel contains two implementations, which we refer to as *TTB_reorder* and

TTB. *TTB_reorder* explicitly reorders the tensor in memory to perform the TTM, whilst *TTB* utilizes the same approach based upon the work of Ballard et al. employed by our implementations. *TTB* was used to generate all of our unit tests to ensure correctness and for runtime comparison for different thread counts.

2.6. InTensLi. InTensLi [10] is an "Input-Adaptive and In-Place Approach to Dense Tensor-Times-Matrix Multiply." Essentially, InTensLi utilizes a novel TTM algorithm to perform the computation in place, saving on the memory footprint of the algorithm. InTensLi utilizes the BLIS library [18] to efficiently calculate strided matrix multiples in order to enable their in-place algorithm. Comparing our CPU runtimes to InTensLi's runtimes on a single node is another potential avenue for future research.

2.7. GPU Distributed Dense Tucker. In 2018 Choi et al. demonstrated a distributed GPU implementation of the Tucker algorithm. An important part of this algorithm was a GPU implementation of TTM. They also proposed efficient ways to access and manipulate the matricized tensor to make it more amenable to GPUs calculations. Analysis contained in Choi et al [3] showing that the TTM kernel takes a significant portion of the ST-HOSVD algorithm further motivates our work.

2.8. Eigen. Eigen [6] is a C++ template library that provides a plethora of highly optimized linear algebra routines including the standard BLAS/LAPACK routines. Eigen is primarily know for its matrix operations, but also provides an optimized suite of various tensor operations. Included in these tensor operations is the tensor contraction operation. The TTM can be considered a special type of tensor contraction wherein one of the tensors is a matrix (2nd order tensor). Eigen can be integrated into CUDA code to run on GPUs, however it depends on using CUDA to explicitly allocate memory. Many projects utilize Eigen, the most notable of which is TensorFlow, the popular machine learning library developed by Google. We compare our TTM implementations against Eigen's tensor contraction function posed as a TTM on three different CPU architectures, showing comparable performance to this state-of-the-art library.

2.9. Modal Product. Golub and Van Loan refer to the TTM as a special form of tensor contraction known as a Modual Product[5]. Van Loan has published several works [15] that have discussed viewing tensor contractions as a series of matrix-matrix multiplications. This is the foundation of the TTM algorithm that the following implementations are based upon.

3. Formal Definition and Notation. The formal definition of the Tensor Times Matrix kernel (TTM), is represented as follows:

DEFINITION 3.1. *Given that $\mathcal{X}$ is a tensor of order N with dimension sizes: $I_0 \times \cdots \times I_{N-1}$ and $\mathcal{U}$ is a matrix of size $J \times I_n$, then $\times_n$ denotes a TTM along the $n'th$ dimension (mode) of $\mathcal{X}$:*

$$\mathcal{Y} = \mathcal{X} \times_n \mathcal{U} \tag{3.1}$$

Where $\mathcal{Y}$ is the resulting tensor with dimensions: $I_0 \dots I_{n-1} \times J \times I_{n+1} \dots I_{N-1}$

The key aspect here is that the tensor's dimension along which the TTM is computed changes to match the number of rows of the input matrix. The TTM can be viewed in terms of matrix multiplication by first *matricizing* the tensor along the given mode. To fully understand tensor matricization it is helpful to first define the concept of tensor *fibers* as follows

DEFINITION 3.2. *Given a tensor $\mathcal{X}$ as previously defined, the mode-n fibers are the set of vectors resulting from holding n'th mode constant and iterating over all other dimensions.*

In other words, the fiber $v_{i_0,\ldots,i_{n-1},i_{n+1},\ldots i_{N-1}} = \mathcal{X}_{i_0,\ldots,i_{n-1},:,i_{n+1},\ldots i_{N-1}}$, *where* : *is used to denote all elements along that dimension.*

The vectors v in Figure 3.1 are the tensor fibers in that example. Based upon this, tensor matricization is defined as follows:

DEFINITION 3.3. *Given a tensor* $\mathcal{X}$, *then* $\mathcal{X}_{(n)}$ *denotes the mode-n matricization of* $\mathcal{X}$ *and is a matrix whose columns are the mode-n tensor fibers of* $\mathcal{X}$ *in column major order. In other words:*

$$x = \sum_{j=0}^{n-1}(\prod_{k=0}^{j} i_k) \rightarrow \mathcal{X}_{(n)}[:,x] = v_{i_0,\ldots,i_{n-1},i_{n+1},\ldots i_{N-1}} = \mathcal{X}_{i_0,\ldots,i_{n-1},:,i_{n+1},\ldots i_{N-1}} \tag{3.2}$$

See Figure 3.1 for the corresponding visual representation. Now the TTM kernel is presented in terms of matrix multiplication and the resulting computational asymptotic complexity is analyzed:

DEFINITION 3.4.

Given $\mathcal{X}, \mathcal{X}_{(n)}, \mathcal{U}$ *and* $\times_n$ *as previously defined, we have that:*

$$\mathcal{Y} = \mathcal{X} \times_n \mathcal{U} \Longleftrightarrow \mathcal{Y}_{(n)} = \mathcal{U}\mathcal{X}_{(n)} \tag{3.3}$$

Defining $I^* = \prod_{r=0}^{N-1} I_r \rightarrow I^{\otimes} = \frac{I^*}{I_n}$, *where* I_r *denotes the size of the r'th dimension of* $\mathcal{X}$, *the resulting asymptotic complexity to calculate* $\mathcal{Y}_{(n)}$ *is:*

$$\mathcal{U} \in \mathbb{R}^{J \times I_n}, \quad \mathcal{X}_{(n)} \in \mathbb{R}^{I_n \times I^{\otimes}} \rightarrow \mathcal{Y}_{(n)} = \mathcal{U}\mathcal{X}_{(n)} \in O(JI_nI^{\otimes})$$

Essentially, the workload of the TTM kernel is equivalent to the product of the dimensions of $\mathcal{X}$ and the first dimension of $\mathcal{U}$.

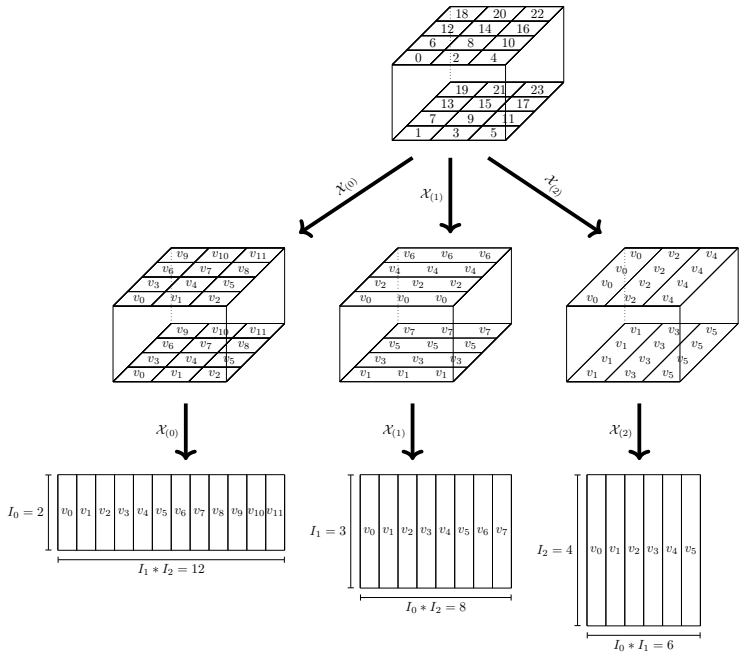


Fig. 3.1: Matricization of $2 \times 3 \times 4$ tensor along each mode

4. Motivation.

4.1. HOSVD. With the formal definition firmly in place attention is directed towards the TTM kernel's primary motivating algorithm, the ST-HOSVD algorithm. The ST-HOSVD has been applied to a wide range of problems that deal with higher dimensional data in fields such as bioinformatics and anomaly detection to make the high dimensional data more amenable to analysis [12] [17]. In addition, the ST-HOSVD algorithm has been extensively used to compresses these high dimensional datasets. This is especially relevant for simulations that are capable of generating many terabytes of data. The aforementioned TuckerMPI [2] applied the ST-HOSVD algorithm to two separate combustion simulation datasets [9] [11], which respectively required 4.4 TB and 6.7 TB to store before compression. In these instances, the ST-HOSVD was shown to have compression ratios of 2×10^5 and 4×10^4 for a relative error bound of $\frac{||\mathcal{X}-\hat{\mathcal{X}}||}{||\mathcal{X}||} < 1 \times 10^{-2}$. Figure 4.1 shows the ST-HOSVD pseudocode and corresponding example.

The running time of the ST-HOSVD algorithm is dominated by the "Gram" matrix kernel (line 3) and the TTM kernel (line 8). As previously discussed, the TTM kernel has an asymptotic complexity of $O(JI_nI^{\otimes})$. Similarly, the Gram matrix kernel, given by:

$$\mathcal{X}_n \in \mathbb{R}^{I_n \times I^{\otimes}}, S \in \mathbb{R}^{I_n \times I_n} \rightarrow S = \mathcal{X}_n \mathcal{X}_n^T \in O(I_n^2 I^{\otimes}) \tag{4.1}$$

Depending on the values of J and I_n relative to each other, the TTM and Gram matrix kernel require asymptotically comparable amounts of work. This bears out in practice, with either of these two kernels dominating the running time of the ST-HOSVD algorithm. Thus, optimizing these two kernels is integral to optimizing the ST-HOSVD algorithm as

Algorithm 1 ST-HOSVD

function ST-HOSVD(Tensor $\mathcal{X}$, accuracy bound ϵ)
  $\mathcal{G} \leftarrow \mathcal{X}$ ▷ Initialize $\mathcal{G}$
  for $n = 0, 1...N-1$ **do**
    $S \leftarrow \mathcal{X}_n \mathcal{X}_n^T$ ▷ Gram matrix
    $[\lambda, V] \leftarrow eig(S)$
    $U_n \leftarrow V(:, 1 : R_n)$ ▷ R_n is smallest value that satisfies ϵ
    $\mathcal{G} \leftarrow \mathcal{G} \times_n U_n^T$ ▷ TTM
  $\mathcal{F} \leftarrow U_0 \dots U_{N-1}$
  return $\mathcal{G}, \mathcal{F}$
end function

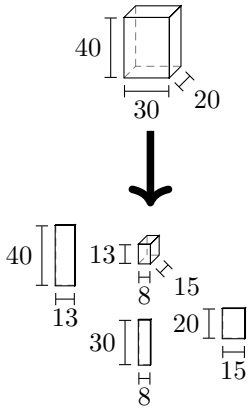


Fig. 4.1: ST-HOSVD pseudocode and example of ST-HOSVD of 3'rd order $40 \times 30 \times 20$ tensor. The small $13 \times 8 \times 15$ tensor in the center of the matrices is referred to as the "core" tensor. The surrounding matrices are generally referred to as "factor" matrices.

a whole. The next section details how the optimized TTM kernel was implemented. The Gram matrix kernel has been developed in parallel to this work.

4.2. Anomaly Detection. We are particularly interested in applying the TTM kernel to calculate the ST-HOSVD of a symmetric 4th order co-kurtosis tensor [16] to facilitate an unsupervised approach for detecting anomalies in combustion simulations. In this application, each dimension represents a set of spatio-temporal features that vary over the course of the simulation. For example, Aditya et al [1] demonstrate a combustion simulation that depends on 12 different chemical species mass fractions and the temperature at each time step. These features can be used to form a $13 \times 13 \times 13 \times 13$ symmetric co-kurtosis tensor that can be decomposed into a ST-HOSVD to detect anomalous events such as abnormal temperatures or chemical ratios. In this instance, we want to calculate the ST-HOSVD in real time and as fast as possible, so that these anomalous events can be acted upon soon after their inception. We thus optimized our TTM implementation to efficiently handle these relatively small symmetric tensors.

5. Implementations. We now move onto the technical implementation of the TTM kernel. Our implementation and discussion thereof is heavily based upon the work of Ballard et al. [2]. Two values that will be helpful in this discussion are:

Definition 5.1.

$$I^{>} = \prod_{r=n+1}^{N-1} I_r, \quad I^{<} = \prod_{r=0}^{n-1} I_r \tag{5.1}$$

Where I, N and n are as previously defined

Assuming the entries of $\mathcal{X}$ are stored in column-major order as a contiguous 1D array, i.e. the mode-0 fibers are contiguous in memory, then the matrix resulting from the matricization of $\mathcal{X}$ consists of $I^{>}$ submatrices that are contiguous in memory, each with I_n rows and $I^{<}$ columns for each $n > 1$. In other words, each matricization not along the first mode consists

of a number of contiguous submatrices equal to the product of the dimensions greater than the mode, where the number of rows of each submatrix is equal to the size of the mode's dimension and the number of columns is equal to the product of the dimensions less than the mode. See Figure 5.1. In the instance that $n = 0$, i.e. the TTM is calculated along mode-0, the matricized tensor consists of $I^{>}$ column vectors. As discussed later, this slightly affects the implementation, making the mode-0 TTM a special case.

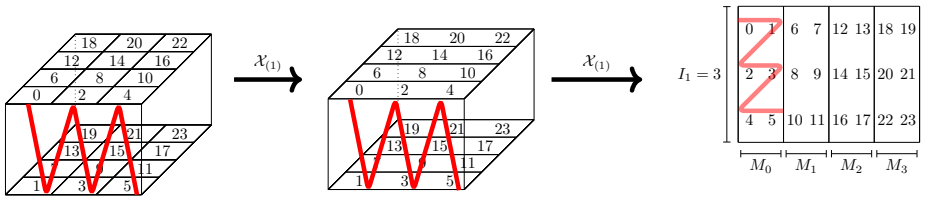


Fig. 5.1: Matricization of $2 \times 3 \times 4$ tensor along mode-1. Notice how the tensor is stored in column-major order, but the submatrix is in row-major order.

In this representation of the matricized tensor, the TTM can be viewed as a series of matrix-matrix multiplications wherein the factor matrix $\mathcal{U}$ is multiplied by the individual submatrices. This has the benefit that no data reordering is necessary if careful attention is given to the memory layout of the submatrices when doing the matrix-matrix multiplication. Specifically, the submatrix must either be stored as row-major, i.e. the fibers of the last dimension are contiguous in memory, or the entire computation must be transposed and the result stored in row-major order. In the mode-0 case this is not necessary because the submatrices are actually vectors (mode-0 fibers) and thus already contiguous in memory. An added benefit of viewing the computation in this manner is that it becomes readily apparent that we may do these submatrix multiplications in parallel. See Figure 5.2.

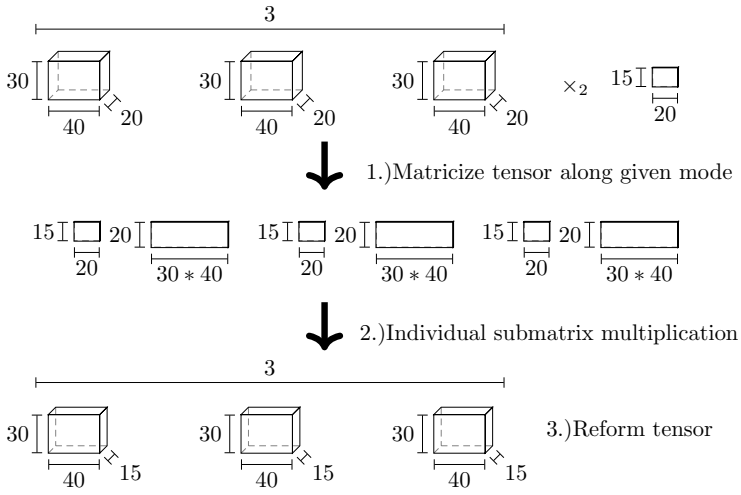


Fig. 5.2: $30 \times 40 \times 20 \times 3$ tensor times 15×20 matrix along mode 2

Taking these arrangements of the submatrices into account, the submatrix multiplications may be done via a call to a general matrix-matrix multiplication (GEMM) kernel.

This grants access to the multitude of highly optimized pre-existing linear algebra libraries, such as BLAS and LAPACK, each of which support general matrix-matrix multiplication. Furthermore, due to the fact that the submatrix multiplications involve separate contiguous sections of the input and output tensors, these GEMM calls can be executed in parallel. Several implementations utilize the system's installed BLAS/LAPACK DGEMM function. In the experiments these took the form of Intel MKL and OpenBlas. Maintaining the contiguousness of the submatrices also allows for the use of the KokkosKernels TeamGEMM function, which is a key component of making a complete Kokkos implementation. Thus, the implementations discussed in subsequent sections fall into two main categories: KokkosKernels implementations and DGEMM implementations. We also implemented a TTM kernel that utilizes CUDA's cuBLAS GEMM function. This combined with the KokkosKernerls implementations will help ensure that our TTM function will perform well on a wide range of accelerator architectures, both present and future. See Algorithm 2 for the general pseudocode of the TTM algorithm.

Algorithm 2 Tensor Times Matrix(TTM). $\mathcal{X}_{(n)}[i]$ denotes the i'th submatrix of $\mathcal{X}_{(n)}$. The format in which $\mathcal{X}_{(n)}[i]$ is stored and whether $\mathcal{X}_{(n)}[i]$ is transposed depends on the implementation.

```
1: function TTM(Tensor 𝒳, 𝒰, n)
2:     [I^<, I^>] ← 𝒳                          ▷ calculate I^< and I^>
3:     𝒳_(n) ← 𝒳                               ▷ matricize 𝒳
4:     if n > 0 then
5:         for i = 0 : (I^> − 1) do            ▷ Change for to parfor in parallel case
6:             𝒴_(n)[i] = 𝒰𝒳_(n)[i]             ▷ done via GEMM call using given library
7:     else
8:         𝒴_(n) = 𝒰𝒳_(n)
9:     𝒴_(n) = 𝒰𝒳_(n)
                    ▷ 𝒴_(n) stored in memory identically to 𝒴, no extra work needed
10:    return 𝒴
11: end function
```

5.1. Kokkos-Kernels. Over a dozen different working Kokkos based TTM implementations were developed in our pursuit of a fully optimized TTM kernel. For the sake of brevity we present only one, *genten_ttm_factor_direct*. In the results section we refer to this implementation as the TeamGemm implementation because it primarily utilizes KokkosKernel's TeamGemm function to do the matrix-submatrix multiplications.

The inputs to the templated *genten_ttm_factor_direct* function are as follows:

- mode: dimension along which to do the TTM
- tensor: tensor is the tensor $\mathcal{X}$ to undergo the TTM. Tensor is of type: TensorT<ExecSpace>. TensorT is a multidimensional type unique to GenTen and is stored as a 1D Kokkos View. The entries of the TensorT are stored in column-major order, i.e. entries in the same column are contiguous in memory. ExecSpace refers to the Kokkos Execution space that the TTM uses. This generally either refers to the host (CPU) execution space or the execution space of a GPU accelerator.
- matrix: predictably matrix refers to the matrix $\mathcal{U}$ to multiply with the tensor. It is of type FacMatrixT<ExecSpace>. FacMatrix is stored as a 2D LayoutRight view.
- ans: ans is the tensor that is used to store the result $\mathcal{Y}$ of the TTM.

Algorithm 3 genten_ttm_factor_direct pseudocode using abbreviated Kokkos syntax.

```
1: function GENTEN_TTM_FACTOR_DIRECT(TensorT 𝒳, FacMatrixT 𝒰, mode n, TensorT 𝒴, team size ts, templated
   execution space ExecSpace)
2:     [I*, I^⊗, I_n, I^<, I^>] ← 𝒳                          ▷ calculate values defined in 3.1, 3.3, 5.1
3:     [J, I_n] ← 𝒰                                                          ▷ dimensions of 𝒰
4:     Kokkos::TeamPolicy< ExecSpace > policy(I^>, ts)          ▷ define Kokkos team execution policy
5:     Kokkos::TeamPolicy< ExecSpace >::member_type member_type
6:     Kokkos::View<ttb_real**, Left, Unmanaged> 𝒰'(𝒰.data(), J, I^n)          ▷ convert 𝒰 to View
7:     if n > 0 then
                                                                              ▷ matricize 𝒳
8:         Kokkos::View<ttb_real**, Right, Unmanaged> 𝒳_(n)(𝒳.data(), I^> * n, I^<)
9:         Kokkos::parallel_for (policy, member_type member) do
10:            i = member.league_rank()
11:            𝒳_(n)[i] = Kokkos::subview(𝒳_(n), iI_n : (i + 1)I_n, Kokkos::ALL())
                                                          ▷ extract i'th submatrix from 𝒳_(n)
12:
13:            𝒴_(n)[i] = Kokkos::subview(𝒴, iJI^< : (i + 1)JI^<)      ▷ extract i'th submatrix from 𝒴_(n)
14:
15:            Kokkos::View<ttb_real**, Right, Unmanaged> 𝒴'_(n)[i] =
                   Kokkos::subview(𝒴_(n)[i],J, I^<) ▷ Recast 𝒴_(n)[i] as 2D LayoutRight unmanaged view
16:
17:            KokkosBatched::Gemm<member_type,'NT','NT',Team,Blocked> →
                   invoke(𝒰',𝒳_(n)[i],𝒴'_(n)[i])      ▷ KokkosKernels TeamGEMM call
18:        end parfor
19:    else
20:        Kokkos::View<ttb_real**, Left, Unmanaged> 𝒳_(n)(𝒳.data(), I^> * n, I^<)          ▷ matricize 𝒳
21:        Kokkos::parallel_for (policy, member_type member) do
22:            i = member.league_rank()
23:            𝒳_(n)[i] = Kokkos::subview(𝒳_(n), Kokkos::ALL(), i : (i + 1))      ▷ extract i'th vector from 𝒳_(n)
24:            𝒴_(n)[i] = Kokkos::subview(𝒴, iJI^< : (i + 1)JI^<)      ▷ extract i'th vector from 𝒴_(n)
                                                                  ▷ KokkosKernels TeamGEMM call
25:            KokkosBatched::Gemm<member_type,'NT','NT',Team,Blocked>::invoke(𝒰',𝒳_(n)[i],𝒴_(n)[i])
26:        end parfor
27:    return 𝒴
28: end function
```

The *genten_ttm_factor_direct* function starts by calculating $I^*, I^{\otimes}, I_n, I^<$ and $I^>$ as seen in Algorithm 3. The team execution policy is then defined to iterate over each submatrix in parallel, using the Kokkos::AUTO() function to determine how many threads to devote to a team. If the mode is not 0, Line 7 then matricizes the tensor by casting it as a 2D LayoutRight unmanaged view. Note that because the $\mathcal{X}_{(n)}$ is an unmanaged Kokkos view, no explicit data copy is required. In Kokkos syntax, the LayoutRight view is stored in row-major order, whilst a LayoutLeft view is stored in column-major order. Thus casting the view in this manner and setting the 2D view dimensions to $I^> * n, I^<$ effectively transposes the submatrices. Line 8 initiates the Kokkos parallel for-loop in accordance with the aforementioned team policy. Lines 10-13 extract the submatrix memory addresses from the matricized tensor and the corresponding submatrix memory regions from the ans tensor. Finally Line 15 executes the matrix-submatrix multiplication, using team_size threads to do so, and writes the result out into the ans tensor. The mode 0 case requires the same process with the simplification that no transposition is necessary for the input or output tensor submatrices.

The KokkosKernels TeamGemm was originally developed for use in the Sandia Parallel Aerodynamics and Reentry Code (SPARC) [7] project where its primary purpose was to operate on small (dimensions less than 32×32) matrices in a Computational Fluid Dynamics (CFD) application. Thus the TeamGemm function that the performance portable Kokkos implementation relies upon is primarily optimized for small matrix-matrix multiplications. This becomes readily apparent in the Performance Results section, as in most cases it outperforms all other implementations for problem sizes where the input matrices are small enough to fit in cache and higher thread counts, but lags behind for the larger problems on

lower thread counts. It is also worth noting that the *genten_ttm_factor_direct* is still under construction. Work is currently underway to develop a simple heuristic to dynamically determine the policy team size based upon the number of available threads and the number of required parfor loop iterations.

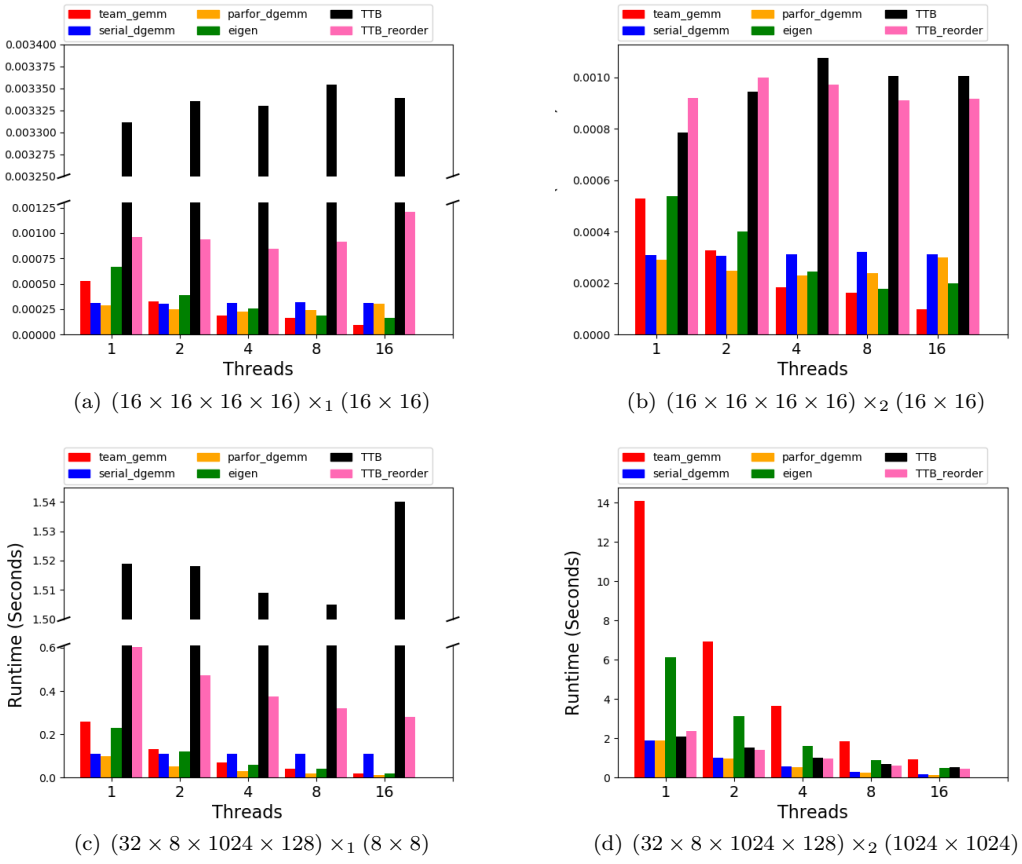
5.2. Default DGEMM. The next two TTM functions that were benchmarked utilize the installed BLAS/LAPACK library of each benchmarked system to facilitate the matrix-submatrix multiplication. The first, *genten_ttm_parfor_dgemm*, is similar to the TeamGemm approach in that it takes advantage of multiple levels of parallelism by utilizing a Kokkos parfor loop and thread parallelism within the DGEMM function itself. It thus represents a form of finer grain parallelism. When benchmarking this function careful attention was given to properly set the OpenMP environment variables to ensure that the function was not using more threads than specified by the benchmark instance. This is a concern due to the hierarchical parallelism of using the an optimized BLAS/LAPACK DGEMM function, which has the potential to automatically use thread parallelism, inside of the Kokkos parallel_for region. The advantage of this method is that generally BLAS/LAPACK libraries are more generalized and thus performs better for larger problem sizes than the specialized KokkosKernerl's TeamGemm function. Conversely, the disadvantage of the *genten_ttm_parfor_dgemm* function is that it is significantly slower than *genten_ttm_factor_direct* for small problem sizes. This is born out in the benchmark results. Additionally, *genten_ttm_parfor_dgemm* does not fully utilize the Kokkos Programming model and thus is not performance portable.

The second DGEMM TTM implementation, *genten_ttm_serial_dgemm*, is almost exactly the same as *genten_ttm_parfor_dgemm* with the exception that the for-loop around the matrix-submatrix multiplication is serial instead of parallel. This function thus derives all its parallelism from the builtin thread parallelism of the linked BLAS/LAPACK DGEMM function.

6. Performance Results and Discussion.

6.1. CPU Results. Timings were generated across Intel Xeon, ARM ThunderX2 and IBM Power9 CPU architectures as well as NVIDIA Kepler and Volta GPU architectures. We present timings from $16 \times 16 \times 16 \times 16$ and $32 \times 8 \times 1024 \times 128$ tensors. The 16^4 instance was motivated by the aforementioned combustion simulation application, whereas the $32 \times 8 \times 1024 \times 128$ was chosen to demonstrate a slightly larger problem size that has a wide range of dimension sizes. As will be shown, this problem size highlights both the strength and weaknesses of our Kokkos based TTM kernel. Interestingly, the comprehensive experiments of these two problems formed a tensor of their own. In total timings were generated for each mode of each tensor for at least four different implementations on each CPU architectures. Not taking varying thread counts into account, this forms a $4 \times 2 \times 4 \times 3$ tensor with 96 entries. The portable Kokkos based implementation capable of running on GPUs was compared with a cuBlas implementation that provided another 32 timing results. Thus, for the sake of brevity we focused on the results along modes 1 and 2 of the chosen tensors and restricted ourselves to a maximum of 16 threads. Modes 1 and 2 are the most interesting because the last mode merely consists of a single GEMM call and the first mode is a series of matrix vector products. For the $16 \times 16 \times 16 \times 16$ problem size, modes 1 and 2 will respectively consist of 16^2 $(16 \times 16) \times (16 \times 16)$ and 16 $(16 \times 16) \times (16 \times 16^2)$ matrix multiplications. Similarly, for the $32 \times 8 \times 1024 \times 128$ tensor, the TTMs along modes 1 and 2 will respectively consist of $1024 * 128$ $(8 \times 8) \times (8 \times 32)$ and 128 $(1024 \times 1024) \times (1024 \times (8 * 32))$ matrix multiplications. Before starting the experiments, we hypothesized that the portable Kokkos based TeamGemm function would perform well when the input matrix was small due

to its originally being optimized for small matrix multiplications. Similarly, we hypothesized that it would not perform well for large input matrices. As we shall see, this is confirmed in the experiments.



(a) $(16 \times 16 \times 16 \times 16) \times_1 (16 \times 16)$

(b) $(16 \times 16 \times 16 \times 16) \times_2 (16 \times 16)$

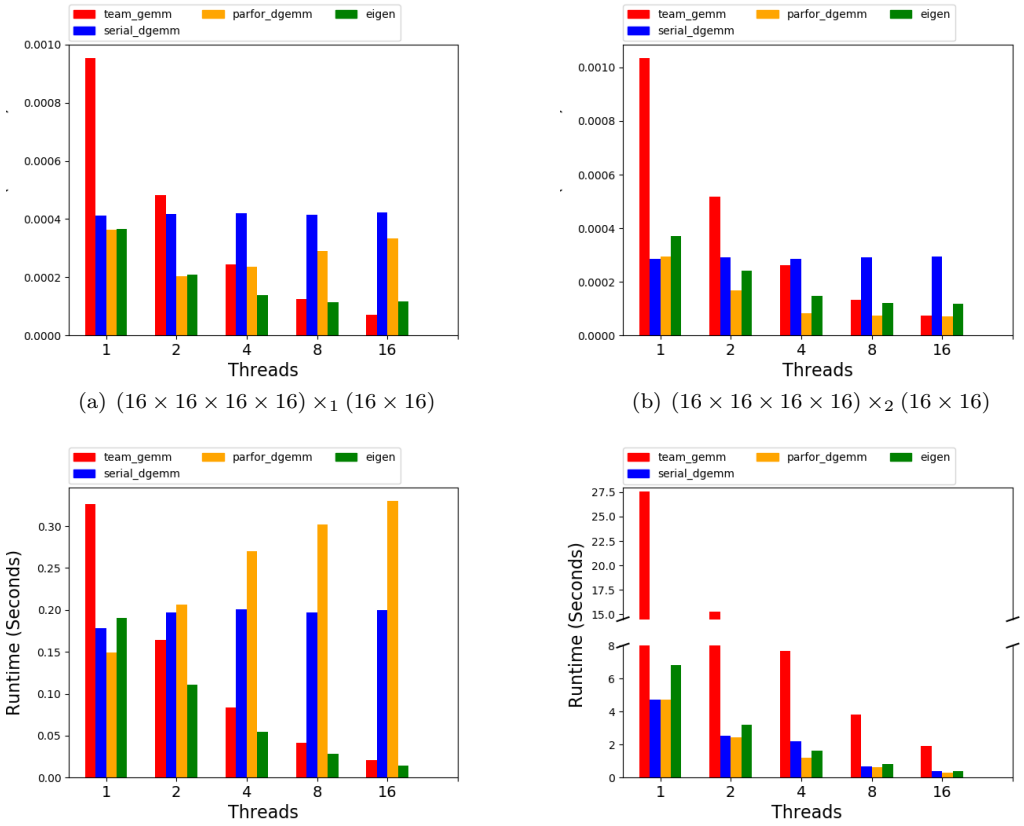
(c) $(32 \times 8 \times 1024 \times 128) \times_1 (8 \times 8)$

(d) $(32 \times 8 \times 1024 \times 128) \times_2 (1024 \times 1024)$

Fig. 6.1: Timings generated on Intel Xeon-2683

Timing results in Figure 6.1 were generated on Kahuna, a heterogeneous Sandia testbed with 120 dual-socket, 28 Intel E5-2683v3 2 GHz core heterogenous nodes. Each core has access to a 32K L1, 256K L2 and 35840K L3 cache. Several nodes are also equipped with Kepler K80 GPUs and one node is equipped with a Volta100 GPU. The Kepler timings were generated using these GPUs. This server also has access to a Matlab license. Thus we were able to benchmark the Matlab Tensor Toolbox in addition to Eigen and the DGEMM implementations for comparison to the portable TeamGemm implementation. In this instance Intel's MKL was the BLAS/LAPACK library that was linked to the GenTen build and provided the DGEMM kernel for the serial and parfor DGEMM implementations. Developed by Intel, MKL performs well on it's native Intel architecture for the larger 1024×1024 input matrix in Figure 6.1(d) compared to both TeamGemm and Eigen. For the problem instances in Figures 6.1(b) and 6.1(c) where the input matrix is much smaller and able to fit in L1 cache, the TeamGemm implementation achieves the fastest runtime for higher thread counts. Figure 6.1(c) demonstrates that the Matlab Tensor Toolbox's TTM function and, in turn, Matlab's built-in GEMM routines performs poorly when the input matrix is small.

In this instance the other implementations performed comparatively better.



(a) $(16 \times 16 \times 16 \times 16) \times_1 (16 \times 16)$

(b) $(16 \times 16 \times 16 \times 16) \times_2 (16 \times 16)$

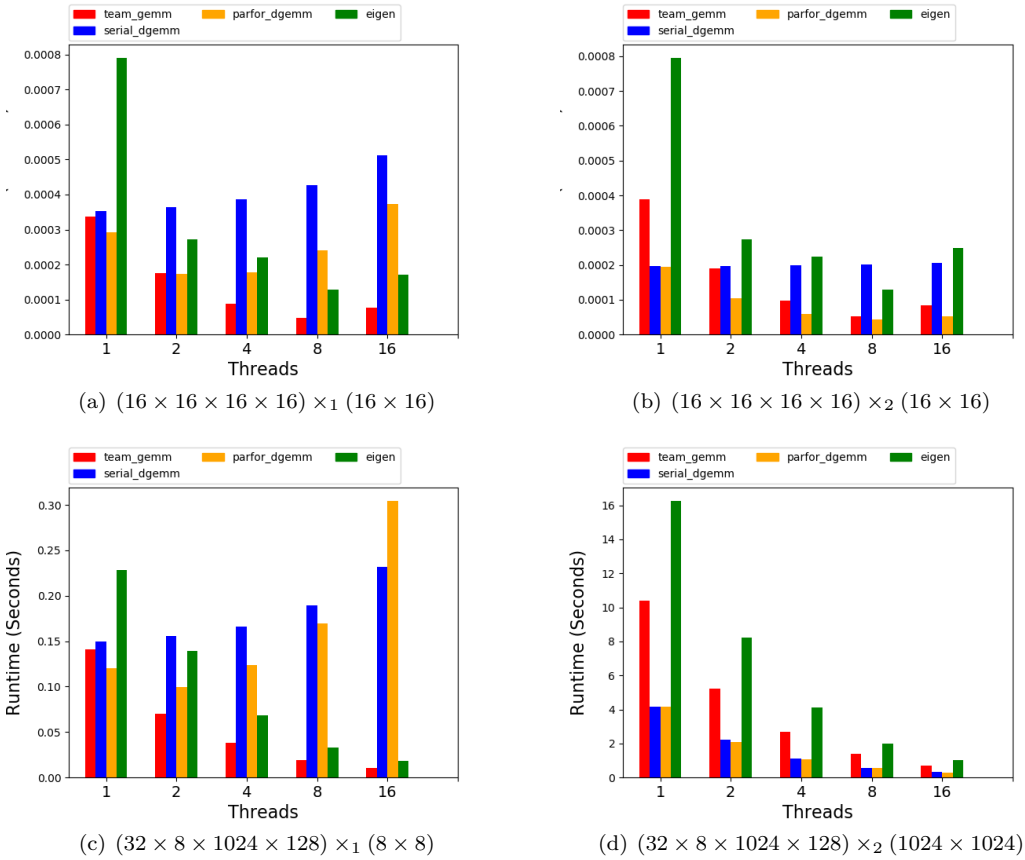
(c) $(32 \times 8 \times 1024 \times 128) \times_1 (8 \times 8)$

(d) $(32 \times 8 \times 1024 \times 128) \times_2 (1024 \times 1024)$

Fig. 6.2: Timings generated on ARM64 ThunderX2

Timing results in Figure 6.2 were generated on Mayer, a Sandia testbed with 40+ dual-socket, 28 ThunderX2 ARM64 core nodes. Each core has access to a 32K L1, 256K L2 and 35840K L3 cache. In this case the DGEMM kernel is provided by OpenBlas, an optimized BLAS library that supports optimizations for both ARM Power9 architectures. As seen in Figure 6.2(d), the DGEMM implementations based upon the OpenBlas kernel and Eigen both scaled quite well for the larger problem size. Unsurprisingly, the TeamGemm implementation did not perform as well for the large input matrix. In all cases the sequential performance of the TeamGemm implementation was slower by at least a factor of two. However, similar to the Intel results, Figures 6.2(a), 6.2(b) and 6.2(c) show the TeamGemm performs well for higher thread counts when the input matrix is small, eventually becoming the fastest on 16 threads for both modes 1 and 2 of the 16^4 tensor. Furthermore, the DGEMM implementations and, in turn, the OpenBlas DGEMM kernel, show worse performance the smaller the input matrix is. From this we can deduce that the OpenBlas GEMM kernel is not sufficiently optimized for instances when at least one of the matrices is small enough to fit in cache.

Timing results in Figure 6.3 were generated on Weaver, a Sandia testbed with 10 dual-socket, 40 core heterogeneous nodes. Each core supports up to 4 threads, allowing a total



(a) $(16 \times 16 \times 16 \times 16) \times_1 (16 \times 16)$

(b) $(16 \times 16 \times 16 \times 16) \times_2 (16 \times 16)$

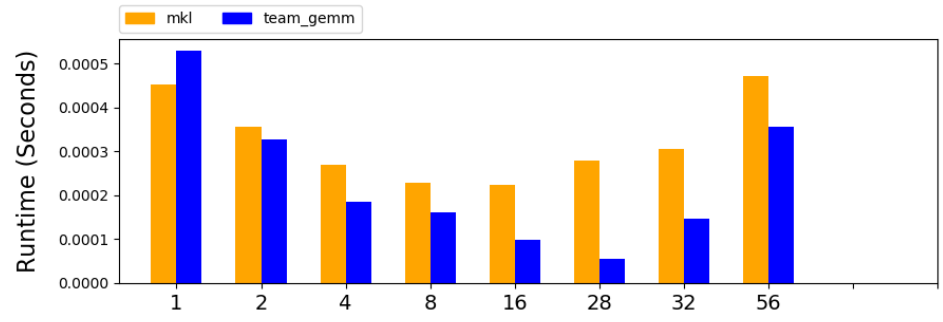
(c) $(32 \times 8 \times 1024 \times 128) \times_1 (8 \times 8)$

(d) $(32 \times 8 \times 1024 \times 128) \times_2 (1024 \times 1024)$

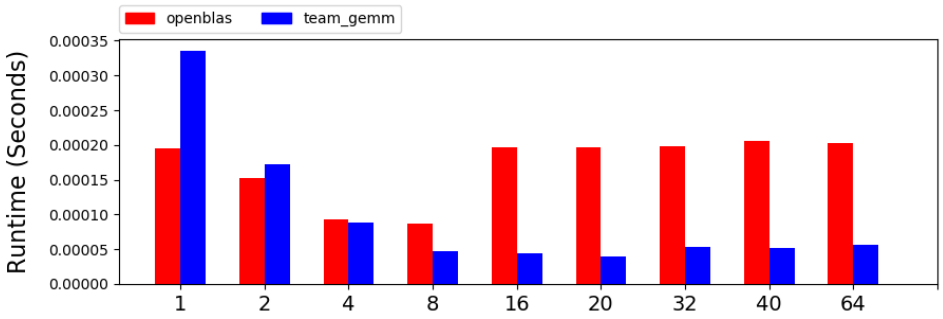
Fig. 6.3: Timings generated on IBM Power9

of 160 threads per node. Each node is also equipped with a Volta100 GPU. The Volta GPU timings for the portable TeamGemm implementation were generated using these GPUs. Previously on the ARM architecture, the TeamGemm implementation lagged behind Eigen for single thread sequential execution. On the IBM Power9 architecture their roles are reversed, with the portable TeamGemm outperforming Eigen in all cases. Similar to the ARM server, the Power 9 server utilizes OpenBlas for its optimized BLAS/LAPACK library. For single threaded sequential execution, the DGEMM implementations that rely upon OpenBlas's DGEMM function outperformed both Eigen and the TeamGemm implementation. However, as seen in Figures 6.3(a), 6.3(b) and 6.3(c), the DGEMM implementations and OpenBlas DGEMM function do not scale well for small input matrices. Based upon the ARM results, this and the fact that the DGEMM implementations scale well for the large input matrix sizes in Figure 6.3(d) is relatively unsurprising.

Figure 6.4 compares the team_gemm TTM implementation to a comparably sized DGEMM call on the aforementioned Intel and IBM servers. The thread counts were increased based upon the core counts of each server in order to demonstrate threads counts higher than 16. As previously mentioned the $(16 \times 16 \times 16 \times 16) \times_1 (16 \times 16)$ will consist of 16^2 $(16 \times 16) \times (16 \times 16)$, which is equivalent to $(16 \times 16) \times (16 \times 16^3)$ in terms of flop count. From this we see that the team_gemm TTM implementation attains DGEMM like per-



(a) $(16 \times 16) \times (16 \times 16^3)$ vs $(16 \times 16 \times 16 \times 16) \times_1 (16 \times 16)$



(b) $(16 \times 16) \times (16 \times 16^3)$ vs $(16 \times 16 \times 16 \times 16) \times_1 (16 \times 16)$

Fig. 6.4: Comparison of TTM vs DGEMM times. MKL benchmarked on Intel Xeon-2683. OpenBlas benchmarked on IBM Power9

formance compared to both MKL and OpenBlas for a co-kurtosis problem size. In both instances, MKL and OpenBlas show degraded performance when the number of threads is greater than the dimensions of the input matrix. On the Power9 architecture the team_gemm implementation does not show significant performance degradation, especially when compared to OpenBlas. The team_gemm implementation shows significantly more performance degradation on the Intel architecture, but still less than MKL. From this we can deduce that neither OpenBlas nor MKL are optimized for our intended problem size. This is consistent with the previous results of the DGEMM TTM implementations that relied upon the OpenBlas and MKL DGEMM kernels.

6.2. GPU Results. GPU results were generated on NVIDIA Volta and Kepler GPUs as seen in Tables 6.1, A.1, A.2 and A.3. Similar to the CPU results the GPU results of the TeamGemm implementation performed comparably to those utilizing the cuBlas function when the input matrix was small, but performed poorly in comparison to the cuBlas implementation for larger input matrices. In this instance the first two dimensions are close to those of our targeted co-kurtosis problem size. The rest of the comprehensive GPU timings are listed in Appendix A.

Volta

Mode	0	1	2	3
TeamGemm	0.0096	0.0014	0.7212	0.0075
cuBlas	0.0126	0.0010	0.0108	0.0017

Table 6.1: $(32 \times 8 \times 1024 \times 128)$

7. Conclusions and Future Work. We are motivated by the need for a fast, performance portable TTM kernel optimized for the problem size encountered in combustion simulations. Several implementations have been compared based upon runtime across several CPU and GPU architectures, demonstrating the performance portability of the KokkosKernels TeamGemm implementation for our intended problem size. Attaining performance portable implementations of both the TTM and Gram matrix kernels is crucial to attaining a ST-HOSVD algorithm with the same performance and characteristics. Such a ST-HOSVD implementation would be an invaluable tool to a wide variety of researchers in fields that deal with high dimensional data, especially those working with many small higher order tensors that arise in combustion simulations.

8. Acknowledgements. We would like to thank Sivasankaran Rajamanickam of Sandia National Laboratories for his invaluable Kokkos advice. It was originally his idea to investigate using the Kokkos TeamGEMM function in our implementations.

REFERENCES

[1] K. Aditya, H. Kolla, W. P. Kegelmeyer, T. M. Shead, J. Ling, and W. L. Davis, *Anomaly detection in scientific data using joint statistical moments*, Journal of Computational Physics, 387 (2019), p. 522–538.

[2] G. Ballard, A. Klinvex, and T. G. Kolda, *Tuckermpi: A parallel C++/MPI software package for large-scale data compression via the tucker tensor decomposition*, CoRR, abs/1901.06043 (2019).

[3] J. Choi, X. Liu, and V. Chakaravarthy, *High-performance dense tucker decomposition on gpu clusters*, in SC18: International Conference for High Performance Computing, Networking, Storage and Analysis, 2018, pp. 543–553.

[4] H. C. Edwards, C. R. Trott, and D. Sunderland, *Kokkos: Enabling manycore performance portability through polymorphic memory access patterns*, Journal of Parallel and Distributed Computing, 74 (2014), pp. 3202 – 3216.

[5] G. H. Golub and C. F. Van Loan, *Matrix Computations*, The Johns Hopkins University Press, third ed., 1996.

[6] G. Guennebaud, B. Jacob, et al., *Eigen v3*. http://eigen.tuxfamily.org, 2010.

[7] M. Howard and S. Arunajatesan, *Sparc v. 8/17/2016, version 00*, 10 2016.

[8] T. G. Kolda and B. W. Bader, *Tensor decompositions and applications*, SIAM review, 51 (2009), pp. 455–500.

[9] H. Kolla, X.-Y. Zhao, J. H. Chen, and N. Swaminathan, *Velocity and reactive scalar dissipation spectra in turbulent premixed flames*, Combustion Science and Technology, 188 (2016), pp. 1424–1439.

[10] J. Li, C. Battaglino, I. Perros, J. Sun, and R. Vuduc, *An input-adaptive and in-place approach to dense tensor-times-matrix multiply*, in SC '15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2015, pp. 1–12.

[11] S. Lyra, B. Wilde, H. Kolla, J. M. Seitzman, T. C. Lieuwen, and J. H. Chen, *Structure of hydrogen-rich transverse jets in a vitiated turbulent flow*, Combustion and Flame, 162 (2014).

[12] L. Omberg, G. Golub, and O. Alter, *A tensor higher-order singular value decomposition for integrative analysis of dna microarray data from different studies*, Proceedings of the National Academy of Sciences, 104 (2007), pp. 18371—-18376.

[13] L. Papageorgiou, P. Eleni, S. Raftopoulou, M. Mantaiou, V. Megalooikonomou, and D. Vlachakis, *Genomic big data hitting the storage bottleneck*, EMBnet.journal, 24 (2018), p. 910.
[14] E. T. Phipps and T. G. Kolda, *Software for sparse tensor decomposition on emerging computing architectures*, CoRR, abs/1809.09175 (2018).
[15] S. Ragnarsson and C. F. V. Loan, *Block tensor unfoldings*, 2011.
[16] T. Shead, H. Kolla, A. Konduri, G. Papoola, W. L. Davis, D. Dunlavy, and K. Reed, *A framework for in-situ anomaly detection in hpc environments.*, (2019).
[17] T. Souza, A. L. Aquino, and D. Gomes, *An online method to detect urban computing outliers via higher-order singular value decomposition*, Sensors (Basel, Switzerland), 19 (2019).
[18] F. G. Van Zee and R. A. van de Geijn, *BLIS: A framework for rapidly instantiating BLAS functionality*, ACM Transactions on Mathematical Software, 41 (2015), pp. 14:1–14:33.

Appendix A. GPU Timing Results. Comprehensive GPU timing results are listed in Table A.1, Table A.2, and Table A.3.

Volta

Mode	0	1	2	3
TeamGemm	9.4E-05	9.2E-05	2.1E-04	8.5E-05
cuBlas	6.8E-05	2.5E-05	1.8E-05	1.4E-05

Table A.1: $(16 \times 16 \times 16 \times 16)$

Kepler

Mode	0	1	2	3
TeamGemm	0.0434	0.0075	3.0010	0.0575
cuBlas	0.049305	0.010967	0.066396	0.010112

Table A.2: $(32 \times 8 \times 1024 \times 128)$

Kepler

Mode	0	1	2	3
TeamGemm	0.00020	0.00011	0.00018	0.00011
cuBlas	0.000151	0.000181	0.000044	0.000026

Table A.3: $(16 \times 16 \times 16 \times 16)$

AN ANALYSIS OF USER EXPERIENCE AND SOFTWARE ENGINEERING IMPROVEMENTS IN A CHARACTERIZATION DATA AND INVENTORY MANAGEMENT SYSTEM FOR RADIATION DETECTORS

ALDEN R. FORD*, DAVID MARTIN†, CHARLES J. GIESELER‡, AND BELKIS CABRERA-PALMER§

Abstract. As part of the suite of catalogs developed by Data Mining, Analysis, and Modeling Cell (DMAMC) members to support the U.S. Department of Homeland Security (DHS) Countering Weapons of Mass Destruction (CWMD) Office mission, the Instrument Characterization Catalog (CharCat), intended for the storage and visualization of characterization data pertaining to radiation detectors used by CWMD test scientists, requires further iteration with respect to user experience and software engineering. CharCat is an online reference platform, presented below are updates to this platform and the reasoning for doing so.

1. Introduction. The Instrument Characterization Catalog (CharCat) is a web application fully developed at Sandia National Laboratories to provide easy access and visualization of radiation detectors data.

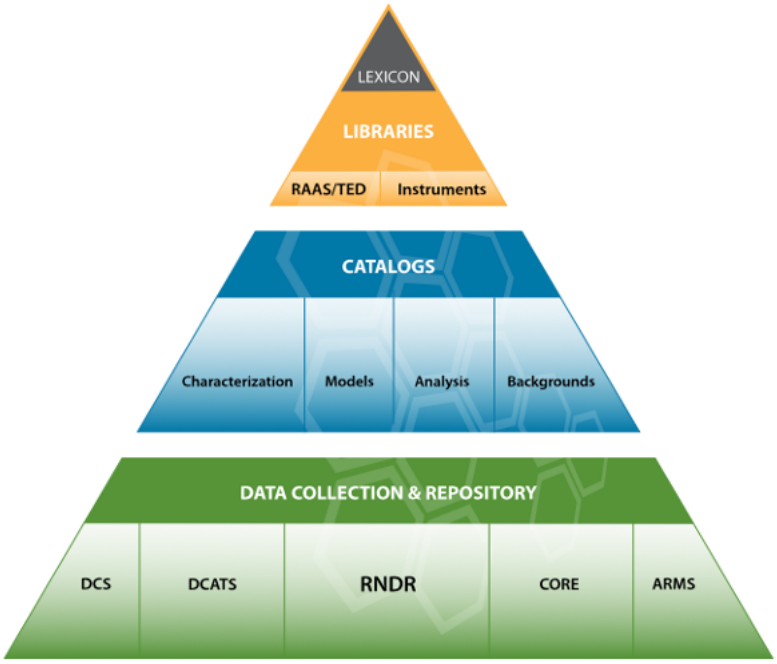


Fig. 1.1: The Instrument **Characterization** Catalog is part of the hierarchy of online applications developed by DMAMC to facilitate the reuse of CWMD collected test data.

As part of the Data Mining, Analysis, and Modeling Cell (DMAMC) suite of catalogs, CharCat supports the U.S. Department of Homeland Security (DHS) Countering Weapons

*Rensselaer Polytechnic Institute, aldenrford@gmail.com
†Arizona State University, dmartin123098@gmail.com
‡Sandia National Laboratories, cjgiese@sandia.gov
§Sandia National Laboratories, bcabrer@sandia.gov

of Mass Destruction (CWMD) Office mission by enabling the reusability of characterization data pertaining to instruments used and tested by CWMD test scientists. As such, the catalog entries are identified by the instrument's unique serial numbers, which permits the application to be also used as an inventory of the CWMD detectors's operational status.

CharCat's initial development phase concluded in fiscal year 2019 (FY19), with the completion of the application's first version and its deployment in an external-facing SNL server. In FY19, CharCat was already conceived with three main components: an inventory home page, the individual detector pages, and a data *wizard* page for new data ingestion. The development work reinitiated in the second half of FY20 has focused on making the application more robust, flexible and user-friendly. Specifically, extensive work has been done to:

- create a framework to easily accommodate changes in data organization,
- refactor the inventory and characterization data ingestion process,
- add exhaustive user guidance features to enhance the users experience.

Biweekly live demonstrations to the CharCat working group, integrated by DMAMC members from the Pacific Northwest National Laboratory (PNNL), the National Institute of Standards and Technology (NIST), the U.S. Naval Research Laboratory (NRL) and CWMD, have served to dynamically absorb their feedback and guide the design and development process.

2. ARF:System Design. CharCat utilizes the MERN (MongoDB, Express.js, React.js, and Node.js)[1] web development tech stack. A principal advantage of the MERN tech stack is that it exclusively uses JavaScript, which allows developers to seamlessly transition between working on client-side and server-side code. In addition, JavaScript provides an efficient way to handle asynchronous behavior without stalling the code through Promises and async/await, an important feature on both the server-side and client-side for an application like CharCat that relies on client-server-database communication chains.

2.1. Front-End. The front-end interface is handled by React.js, a JavaScript library that allows developers to write components, which are reusable modules that can be nested inside other React components. This makes React ideal for handling single-page web applications such as CharCat. As a result, CharCat's front-end is split into a tree of React components that each define a subset of the application's client-side behavior and appearance. Among these, there are three main views - an inventory page that lists instruments, a data wizard that supports uploading data, and an instrument page that displays characterization data (see Figure 2.1). While CharCat is fundamentally a single-page application, it utilizes React Router[2] to provide URLs to access the primary views. CharCat's front-end also makes use of Material UI[3], a library of pre-built React components following Google's Material design principles, to create a visually appealing front-end.

2.2. Server. CharCat utilizes Express.js, a server framework built for the Node.js runtime environment, as its server. Express.js allows developers to define API routes that support CRUD (create, read, update, delete) operations, serving as a bridge between the client and the database. In the case of CharCat, Express.js forms the backbone of how the application stores and displays instrument data by exposing ways to read and manipulate instrument data.

2.3. Database. CharCat uses MongoDB, a popular NoSQL database, to store instrument data. As opposed to SQL databases, NoSQL databases such as MongoDB allow data

[1] mongodb.com, expressjs.com, reactjs.org and nodejs.org
[2] reactrouter.com
[3] material-ui.com

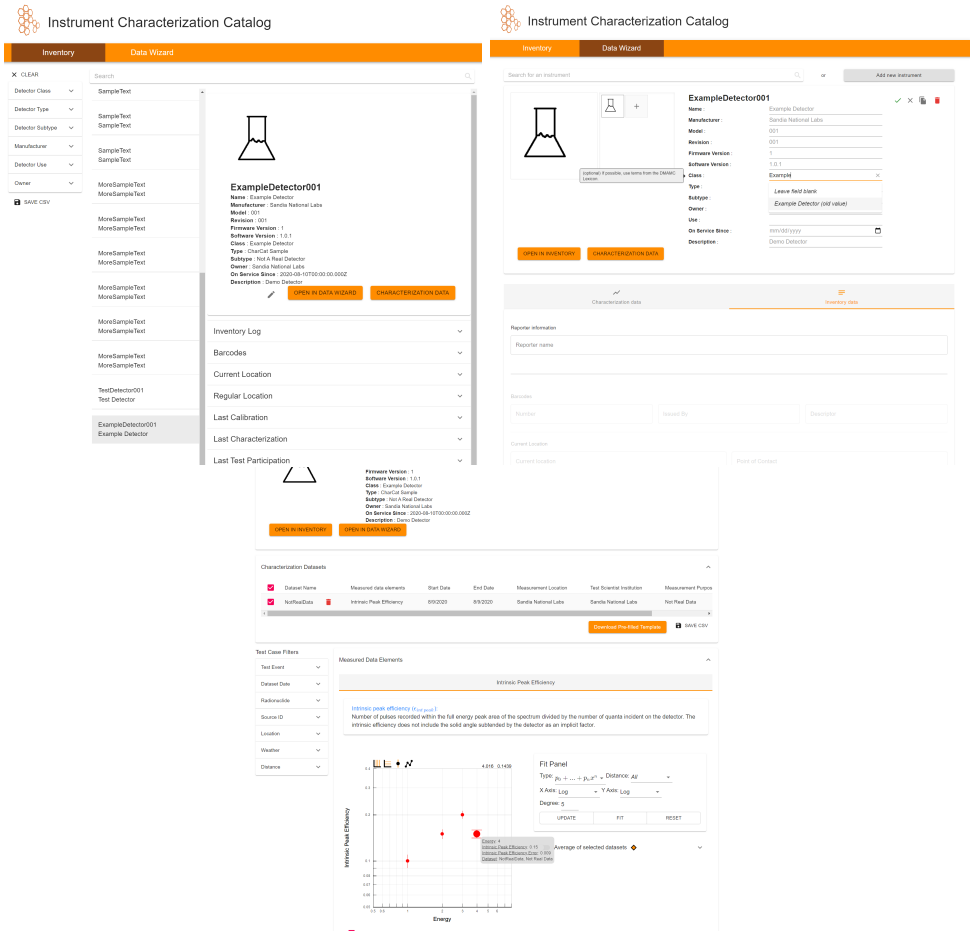


Fig. 2.1: Left-to-right: Inventory page, Data Wizard, Instrument page

to be stored as loose-form JSON as opposed to a fixed-form table. This corresponds well with CharCat's data model (see Figure 2.2), as it allows parent data elements to store an array of references to their child elements. CharCat splits its data into four distinct levels:

- Instruments, which contains inventory data about individual instruments,
- Datasets, which contain metadata about a series of tests performed on an instrument,
- Data elements, which serve serve to group individual test data points by the type of data they represent, and
- Entries, which represent individual data points from tests.

These different levels of data are stored in separate collections (save for data elements, which are stored within the datasets collection as a means to group the dataset-entries relationship). In addition, CharCat also has a "metadatas" collection to store metadata information used in rendering the client, such as the various tooltips present on the client, and an "images" collection used to store the instrument images.

In order to ensure that all the data being stored is in a format that's compatible with

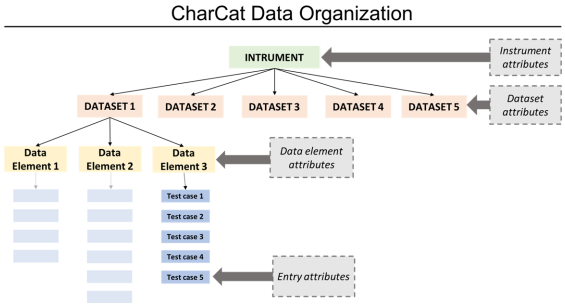


Fig. 2.2: CharCat splits its data into a tiered tree.

CharCat, CharCat uses Mongoose[4] to enforce specific schema upon all entries into the database. A major focus for this period of performance was updating these schemas, which was accomplished with help from migrate-mongo[5] to ensure that all data already in the database was updated to match the modern schema.

2.3.1. Database Image Collection. In terms of the back-end functionality of the Image Gallery component and potential future use cases for images on the front-end, it's also important to address specifically how MongoDB is storing and organizing this data. Under the CharCat database, another collection called "images" is created for the sole purpose of storing image data. The schema of entries to this collection follow the model of first, image buffer data; second, array of serial numbers; and third, image type (such as jpeg, png, etc.). In effort to streamline the design of this collection, another important constraint is the idea that the collection will only store unique images. Leveraging this and the unique identifier tag that is associated with MongoDB entries, it is possible to create, delete, and edit entries on the basis of the image url alone, since it is unique to the collection by design. Now, with the help of ExpressJS, the database collection can be updated to associate a new serial number with a unique image, to removing the image altogether. Specifically, the options presented to the user in the Image Gallery component designate the possible actions that can be taken when updating the collection.

In terms of how the image url is encoded, images are first stripped of their type and then converted to buffer data for storage. When retrieved, they are converted back into a base64 format with their type appended to the front of the string for displaying on the browser. As compared to how images were stored and retrieved in past versions of CharCat, this process represents both a time and space complexity improvement. Previously, images were stored on the server's filesystem and requests were made to retrieve images from these folders on the server. In terms of scalability and security, by storing images as binary data on the database, the application is now more isolated from the server (in the case file permissions change, etc.) and therefore more portable for transporting to other servers for hosting. This is important for the client side as well as the retrieval and storage process is quicker and in line with how other instrument data is stored on the database, making development easier.

3. ARF:User Experience. Because CharCat's role is as an interface to ingest, store, and provide data to scientists, it is important to optimize its user experience. A major component of this period of performance was updating the data ingestion interface to meet this

[4] mongoosejs.com
[5] npmjs.com/package/migrate-mongo

requirement, including adding user assistance features and redesigning several key elements of the interface.

3.1. Tooltips. One essential usability addition is CharCat's tooltips. Due to the technical nature of CharCat's data fields, tooltips provide CharCat with an unobtrusive way to provide the user with additional guidance, containing text ranging from static descriptions of data fields to dynamic explanations of errors. For tooltips present in the front-end web application, CharCat uses Material UI tooltips with custom styling, while for tooltips for Excel templates (explained in Section 3.4) utilize Excel comments.

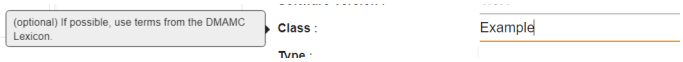


Fig. 3.1: Data Wizard tooltips provide additional information when the user hovers over a field.

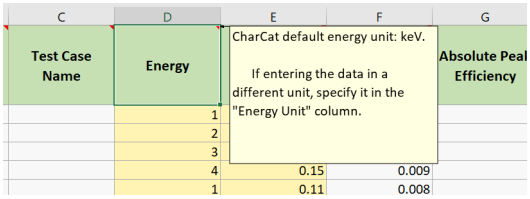


Fig. 3.2: Tooltips present in Excel provide additional information about important columns.

3.2. Inventory Data. In addition to the aforementioned tooltips, the development team added a number of additional features to improve the inventory data user experience.

One such enhancement was the introduction of navigation buttons that allow a user to switch between the Inventory, Data Wizard, and Instrument pages while preserving the selected instrument (see Figure 3.3. This represents a major efficiency improvement for this common task, as the previous workflow to toggle between these views required clicking "Data Wizard" or "Inventory" and searching/scrolling to the desired instrument.

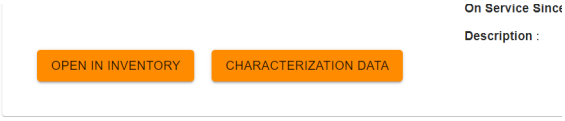


Fig. 3.3: Navigation buttons present in the instrument header allow users to quickly swap between app views while keeping an instrument selected.

The "Data Wizard" view experienced a significant redesign. Both the instrument header and inventory data forms now utilize Material UI's autocomplete component to add a dropdown list of suggested options to a number of text fields (see Figure 3.4). These options are generated based on data already in the database, which allows the user to fill out fields faster such as "Current Location" and "Owner" if the text they want is already present.

This period of performance also introduced a number of error-checking features to the inventory data and instrument header forms. If the user enters text into a field that's

Fig. 3.4: Autocomplete allows the user to quickly select an option for a text field already present in the database or enter their own custom text.

not currently in the database but similar to text that is in the database (calculated with a Levenshtein Distance algorithm), the form will display a warning (see Figure 3.5). In addition, this period of performance also introduced summary pages to the instrument header and inventory data form that appear right before a user submits the form (see Figure 3.6). These summary pages contain a list of changes and display any warnings, providing the user with the opportunity to correct errors before submitting the form.

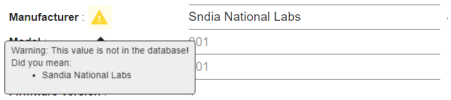


Fig. 3.5: Error warnings allow the user to catch common errors such as typos.

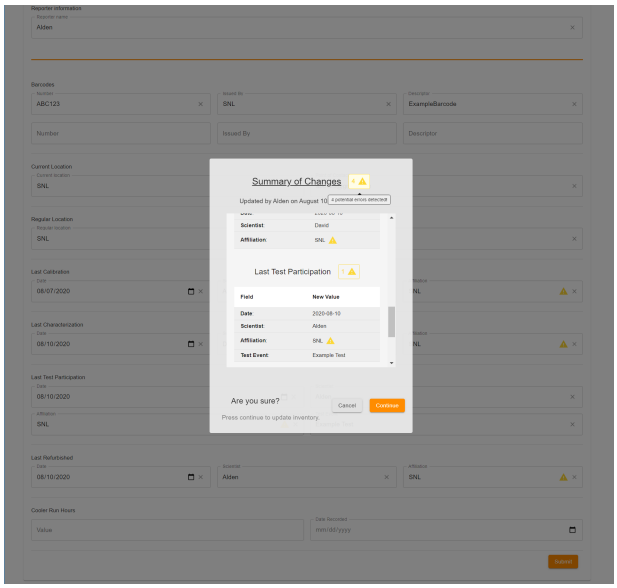


Fig. 3.6: The summary page allows users to review their changes before saving them.

3.3. Image Gallery. Another important component added in this iteration of CharCat is the Image Gallery component. Images represent an effective piece of information for communicating data better in some respects than other mediums. In order to streamline

the process of adding, deleting, and editing images on the application, the development team found it useful to isolate this behavior from the client filesystem as much as possible. For example, given a detector model is added two or more times to the database, of differing serial numbers, then it's natural to assume that an identical image could be used to represent this detector. The cost to the user of uploading this image from the filesystem increases multiplicatively under the current section process for each identical model added to the database. In effort to reduce this time complexity in the improved selection process, it would be useful to maintain a set of all unique images ever uploaded to and associated with a detector in the database, and to present a clear tool for the user to draw these images from in the case another identical detector model is added. This is the idea behind the Image Gallery component.

The component itself is split into the option to add images from the client filesystem and the option to access the global set of unique images. If choosing the latter, the user is presented with a dual layer, searchable component. The first layer acts as an overview displaying thumbnails of unique images and the second, more descriptive layer, shows all models and serial numbers associated with these thumbnail images. In effect, the component acts as a front-end portal into the database collection which stores the images. On top of this, certain features, such as automatic highlighting of currently associated instruments and shading of tags, serves to improve the user experience.

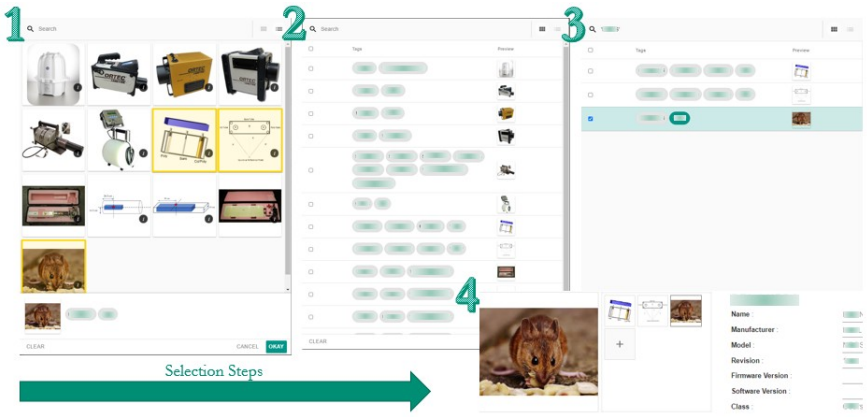


Fig. 3.7: A visual of the image gallery overview/description layers and the image selection process.

3.4. Characterization Data. The characterization data ingestion process saw a significant revision during this period of performance. In addition to general software engineering improvements described in Section 4, the user experience is now significantly more efficient. CharCat uses Excel templates filled out by the user for characterization data ingestion, so these changes primarily focus on improving this aspect of the user experience.

First and foremost, CharCat now allows the user to define Excel templates for instruments rather than relying on instruments already in the database. Prior to this period of performance, instrument templates were manually assembled and manually associated with instruments without a client-side interface. Now, the user defines a list of valid data elements for an instrument through a client-side drop-down menu (see Figure 3.8) and an

Excel template is dynamically generated with the appropriate worksheets using ExcelJS[6].

Fig. 3.8: A Material UI multiple-select autocomplete component allows users to define an instrument's Excel template.

The worksheets now include additional user assistance features. These include an instruction sheet (see Figure 3.9), drop-down menus (Figure 3.10), tooltips (Figure 3.2), and in-sheet data validation (Figure 3.11). These features serve to ensure that the test scientists are able to fill out the Excel templates correctly and avoid server-side validation headaches.

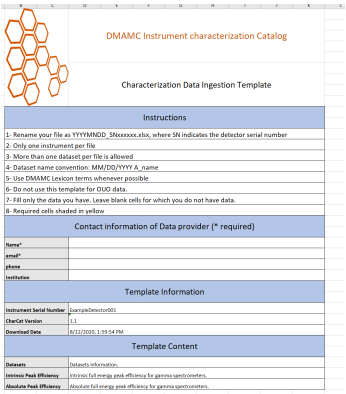


Fig. 3.9: An instruction sheet explains how to use the Excel template.

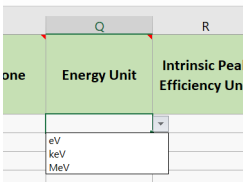


Fig. 3.10: Certain Excel fields have a drop-down of valid options.

The Excel upload process has also been significantly revised. When the user uploads a template, CharCat's server performs validation on the data. If there are any errors not caught by the template (such as using an outdated template version or missing a required field), CharCat provides the user with a summary page of all error messages (see Figure 3.12), allowing the user to find and address the problematic fields. At upload time, CharCat

[6] npmjs.com/package/exceljs

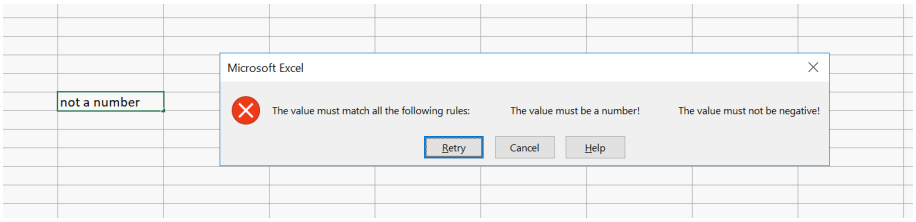


Fig. 3.11: Entering invalid data triggers an error message.

also now converts all data in certain columns to a default unit standardized across datasets, which allows users to submit data with alternate units that their data might be recorded in.

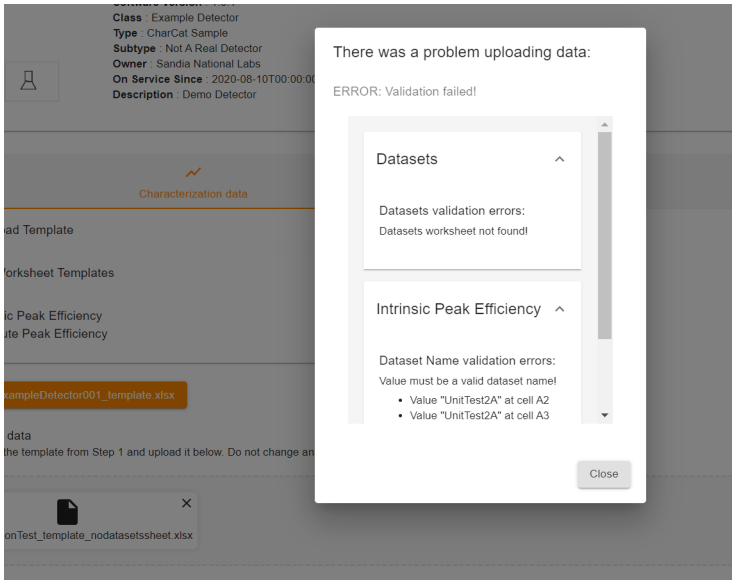


Fig. 3.12: Validation errors are displayed to the user in a summary page.

CharCat now supports the full set of CRUD (create, read, update, delete) operations for datasets. If the user wishes to modify an existing dataset, they can download one or more datasets for an instrument as a prefilled template through the "Instrument" page (see Figure 3.13). Reuploading a dataset already in the database will overwrite the old dataset, allowing users to modify a prefilled template and upload it in order to fix errors in a dataset. CharCat also allows users to delete datasets through a "delete dataset" button.

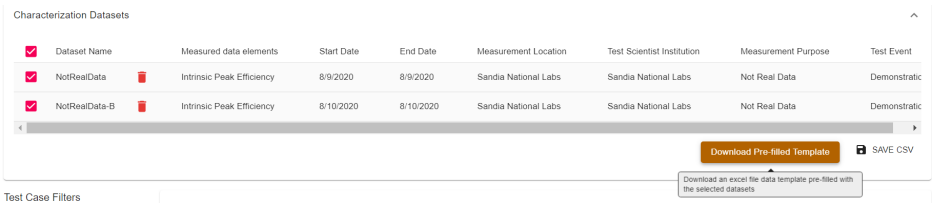


Fig. 3.13: The instrument page allows users to delete datasets as well as download and edit them.

4. Software Engineering. This period of performance also saw CharCat adopt a greater focus on software engineering best practices as it moved away from its prototype stage. While software engineering best practices are helpful for any software engineering project, CharCat requires an extra focus on maintainability because its periods of performance are separated by many months and feature a mostly different team each time.

4.1. Clean Code and Refactoring. In order to accomplish this goal, large sections of CharCat's code were refactored to adopt principles of clean code, such as descriptive function and variable names, comments explaining the purpose of code and how to use it, and "Don't Repeat Yourself". A major improvement was cutting down the amount of repeated code dramatically. For instance, the instrument header component used to be split into two components for its editable and uneditable variants. This meant that any changes to the instrument header had to be repeated in two locations. By merging the instrument header, any changes only need to be made once, dramatically improving ease of maintenance. This sort of consolidation was repeated in a number of places throughout the inventory and characterization ingestion processes, as elaborated in Section 4.2. In addition, many React components were refactored into numerous specialized components, a process designed to significantly reduce the length of individual files. Excessively large source files made it difficult for developers to understand legacy code, so this improvement makes the code easier to maintain.

4.2. Data-driven Design. As part of the "Don't Repeat Yourself" paradigm, many of the refactored components utilize a data-driven design structure. For these aspects of CharCat, their important information is stored as a large JavaScript object, which is then parsed by functions to accomplish tasks such as rendering the client and processing a form.

A major example of this is the "Inventory Data" component. All metadata regarding the inventory is stored in one unified file, such as the layout of the fields, what fields they correspond to in the database, what type of data is stored in the field, the tooltip associated with the field, etc. This data is utilized in the inventory data detail, the inventory log, and the inventory data form. As a result, to change the structure of inventory data, you just need to change one section of the "Inventory Field Data" file rather than update completely different code in three separate locations. In a similar manner, Excel templates are controlled through their own data object. This object defines how to generate Excel templates and also defines how to parse Excel templates. Previous versions of CharCat relied on a premade Excel template and a prototype Python file to parse the Excel templates, which made it incredibly time-consuming to update the Excel templates. Now, an Excel template can be updated by just changing a small segment of code in a JavaScript object.

Aside from simplifying the process of editing layout and templates, data-driven design also dramatically reduces the size of functions required to accomplish tasks. For instance,

while the template data files are several thousand lines long, they're parsed by only a few hundred lines of functions. Smaller functions like these are significantly easier to write, edit, and test because their behavior is more concise.

5. Technology Research. In the fashion of web applications, continuous improvement in order to maintain a healthy ecosystem is the current trend. In light of this, the ability to quantize and measure website traffic is increasingly important. The most common approaches to monitoring and updating a web application rely on the existence of a user management and user activity tracking system. With the use of these technologies, "CharCat" will be able to monitor application behavior and designate user privileges going forward throughout its lifetime.

So what are these technologies? Briefly, user management systems are used to control access to system resources. For example, a user management system is important in the areas of password resets, deleting, creating, or blocking users, and gathering user data. Specifically, CharCat may find a user management system useful for delineating administrative privileges to maintain client features such as access to and deletion of images and instrument data. On the other hand, user activity tracking is the monitoring of user behavior as defined under a user management system. Monitoring behavior could pertain to key-logging, network packet inspection, recording user activity, or analyzing user error logs. It is useful for quantifying actions on the application as malicious or permissible, discovering trends through data analysis, or tracing steps backwards from a discovered bug or error. In a way, user activity tracking allows for continuous development by making a record of each iteration of beta testing of the application and pinpointing important problems to address.

Thanks in large part to the ecosystem behind the NodeJS framework, there exist many software packages available for download. In terms of creating a user management system, user authentication is an essential building block. PassportJS is a useful middleware intended for implementation in ExpressJS that when combined with MongoDB serves this purpose. Simple, but effective, this combination provides a user management system to build a user activity tracking system upon. From here, another software package named Node-Analytics opens CharCat into the realm of user monitoring. This is useful for tracking page visits, user events, and other useful analytics information.

From researching these technologies, there exist an almost infinite array of solutions to generating these systems for use in CharCat. In order to limit scope, software packages should work seamlessly into the MERN stack, be actively maintained, as well as present themselves as a lightweight approach.

6. Conclusions. As a component of DMAMC's suite of applications, it is important that CharCat has an efficient, user-friendly interface and can be maintained and adapted to handle the challenges and requirements posed by its role. In this period of performance, we successfully redesigned the data ingestion user experience to be more user-friendly and robust, improved the maintainability of the source code by applying software engineering best practices, and laid the technology research groundwork for future enhancements.

VERIFYING QUANTUM CIRCUIT EQUIVALENCES USING PROVE-IT, AN INTERACTIVE THEOREM PROVING ASSISTANT

JOAQUÍN E. MADRID LARRAÑAGA** AND WAYNE M. WITZEL††

Abstract. It is challenging to ensure that a quantum program is a faithful implementation of a quantum algorithm after it is tailored and optimized for specific quantum hardware. While quantum computation literature often focuses on the general problem of proving that two arbitrary quantum programs are equivalent, which has exponential complexity with respect to the number of qubits, we consider the simpler, practical problem of verifying each step in a series of small transformations to certify that a final program is equivalent to the original one. We use quantum circuits as an intuitive visual representation of quantum programs and extend Prove-It (a Python-based, Sandia-developed interactive theorem prover) with quantum circuit proof capabilities. This tool could be interfaced with an automated quantum compiler/optimizer or be used interactively by a quantum information expert for exploratory purposes. Using a small number of basic facts about circuit equivalences, Prove-It will be able to solve the verification problem for most practical purposes with linear time complexity.

1. Background. Just as classical computation, at a fundamental level, can be understood as a sequence of logical gate operations acting on classical bits (zeros and ones), quantum computation is understood as a sequence of logical gate operations acting on quantum bits (qubits). For example, a Pauli X gate is a generalization of a classical NOT gate, both of which will transform an input state of "zero" to an output state of "one" and vice-versa. A common way of representing these logic gates within a larger quantum algorithm is with a quantum circuit. A quantum circuit is a visual representation of a time sequence of quantum operations that is performed on one or more qubits. It shows the logic gate operations acting on the qubits in a specific sequence; rows represent qubits and columns represent time that is ordered going left to right (see Figure 1.1).

```
Circuit(ExprArray(ExprTuple(Input(ket_psi), IdentityOp(), IdentityOp(), MultiQubitGate(CONTROL, Set(one, two)),
                            Gate(H), MEAS, MultiQubitGate(CLASSICAL_CONTROL, Set(one, three)), SPACE),
                  ExprTuple(Input(ket0), IdentityOp(), MultiQubitGate(X, Set(two, three)).withStyles(
                                                                        representation='implicit'),
                            MultiQubitGate(X, Set(one, two)).withStyles(representation='implicit'),
                            IdentityOp(), MultiQubitGate(Literal('MEAS'), Set(two, three)), SPACE, SPACE),
                  ExprTuple(Input(ket0), Gate(H), MultiQubitGate(CONTROL, Set(two, three)), IdentityOp(), IdentityOp(),
                            MultiQubitGate(X, Set(two, three)), MultiQubitGate(Z, Set(one, three)), Output(ket_psi)))
       ).withStyles(spacing='@C=3em @R=2.5em')
```

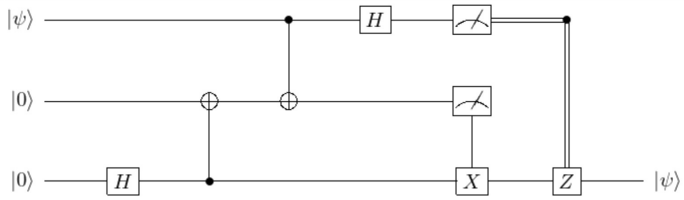


Fig. 1.1: An example of a quantum circuit generated using Prove-It. A quantum circuit is a visual representation of a quantum algorithm, in this case, a quantum teleportation algorithm. Being displayed are input/output cells from executing interactive Python within a Jupyter notebook using the Prove-It library with our quantum circuit extensions.

When an abstract quantum algorithm is implemented on specific quantum hardware, it must be compiled into an implementation that respects the specific hardware limitations

**Occidental College, jmadridlarra@oxy.edu
††Sandia National Laboratories, wwitzel@sandia.gov

and may be optimized for the best performance. Accordingly, one must perform a series of transformations going from an initial quantum circuit that is a literal translation of the algorithm into a final quantum circuit that is tailored to the specific hardware. Various tools exist that perform compilation and optimization for such purposes [12, 11, 13]. How can one be assured, however, that the output of such software is a correct implementation of the circuit that was provided as the input? There is active research [4, 10] to address the challenging problem of blind circuit equivalence verification, trying to determine whether two arbitrary circuits are equivalent. This is a hard problem in general, suffering from the exponential scaling of quantum states with respect to the number of qubits. Some of the research [2, 3] is restricted to the stabilizer formalism only applicable to quantum circuits that can be simulated efficiently by classical computers. Alternatively, we propose to verify each of the basic transformations of a series of compilation/optimization steps, concatenating the derivation to construct a full proof of quantum circuit equivalence. That is, instead of treating the problem in a blind sense, we consider the simpler problem of guided quantum circuit equivalence verification.

In this article, we show progress in developing a tool for this purpose. This tool could eventually be incorporated into compilation/optimization tools so they generate certificates to verify equivalence between the input and output quantum circuits. Verified optimization on arbitrary-sized circuits using symbolic reasoning has been previously demonstrated [9]; however, our tool is unique in allowing users to work directly with convenient, intuitive mathematical notation while verifying and certifying quantum circuit equivalence. It therefore has potential to be particularly valuable for human-guided quantum circuit optimization as well as theoretical explorations over the space of quantum algorithms and implementations.

2. Prove-It. Prove-It is a python-based interactive theorem proving assistant being developed at Sandia National Laboratories in collaboration with the University of New Mexico [14]. The overall aim of Prove-It is to provide a tool for users to prove statements with the same level of ease as an informal proof done by hand. While this paper will not delve into the intricacies of Prove-It as a whole, there are a couple of over-arching principles of Prove-It that are important to discuss.

First, Prove-It is an open source software designed to allow users to add axioms and prove theorems of their own choosing. Theorems are created from expressions formatted in LaTeX as convenient mathematical notation. There is an important distinction between simply creating an expression and proving that this expression is a true statement. Prove-It will allow a user to create any type of expression, and even use it as an assumption, regardless of whether or not it is provable. Some expressions may be proven automatically by calling the ".prove()" method and listing a set of assumptions. In nontrivial instances, the user will need to guide the proof generation through a sequence of derivation steps. As a convenience, theorems may be added and used in the system as a "conjecture," allowing them to be used to prove other statements; these will also be marked as "conjecture" until all dependent theorems have been proven down to the axiomatic level. In order to help users determine whether or not a proof is valid, Prove-It generates an intuitive, easy to read proof, serving as a certificate, that is check-able by both humans and computers. With this in mind, we can easily incorporate and express quantum circuits and related theorems thanks to Prove-It's flexibility.

Prove-It utilizes Python's object-oriented programming ability to create a convenient notation using common mathematical operators such as Add(), InSet(), and Exists(), which take in the user's arguments to create the desired expressions. Using this convention, users can create any type of new operation, implement the Prove-It specific methods, and integrate the new operation into Prove-It's library. The LaTeX output for these new operations may

be implemented freely to generate new mathematical notation at will. By importing the \QCircuit LaTeX package [6], we are able to generate expressions that use quantum circuit notation without changing any of the core Prove-It utilities or the manner in which proof certificates may be confirmed.

Lastly, Prove-It's unique "expression range" (ExprRange) feature allows us to create and instantiate operations with arbitrary, non-specific arity. An ExprRange is an indexed iteration designed to mimic the "..." commonly used in informal proofs to denote a collection of items of variable size. In Prove-It, an ExprRange is of the form $a_m, \ldots, a_n$, where a, m, and n can be defined to create another indexed iteration (Figure 2.1(a)), instantiated to be a specific list (Figure 2.1(b)), or some combination of both. We can use ExprRanges in any context, including new notations created by a user. ExprRanges prove to be integral to the creation of a general quantum circuit theorem that can be instantiated to a specific quantum circuit.

```
tuple_len
```

$$\vdash \forall_{i \in \mathbb{N}}\ [\forall_{a_1,\ldots,a_i}\ (|\,(a_1,\ldots,a_i)\,| = i)]$$

```
tuple_len.instantiate({a:b, i:k},
                      assumptions=
                      [InSet(k, Naturals)])
```

$$\{k \in \mathbb{N}\} \vdash |\,(b_1,\ldots,b_k)\,| = k$$

(a) Instantiating with another generic range

```
tuple_len
```

$$\vdash \forall_{i \in \mathbb{N}}\ [\forall_{a_1,\ldots,a_i}\ (|\,(a_1,\ldots,a_i)\,| = i)]$$

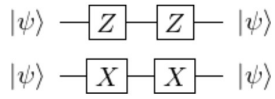
```
tuple_len.instantiate({a:ExprTuple(one, two,
                                   A, B),
                       i:four})
```

$$\vdash |\,(1, 2, A, B)\,| = 4$$

(b) Instantiating a specific list

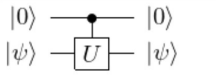
Fig. 2.1: Instantiating "tuple_len" theorem from the `proveit.core_expr_types` theory package to illustrate the flexibility of Prove-It's ExprRanges. Being displayed are input/output cells from executing interactive Python within a Jupyter notebook using the Prove-It library.

```
Circuit(
    ExprArray(
        ExprTuple(Input(ket_psi), Gate(Z),
                  Gate(Z), Output(ket_psi)),
        ExprTuple(Input(ket_psi), Gate(X),
                  Gate(X), Output(ket_psi))))
```



(a) Null circuits created by doubling the gate. Null because the second gate reverses the change made by the first gate

```
Circuit(
    ExprArray(
        ExprTuple(Input(ket0),
                  MultiQubitGate(CONTROL,
                                 Set(one, two)),
                  Output(ket0)),
        ExprTuple(Input(ket_psi),
                  MultiQubitGate(U, Set(one, two)),
                  Output(ket_psi))))
```



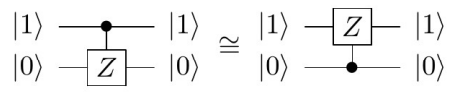
(b) A control gate needs an input of 1 to have any effect so an input of 0 nullifies the control gate

Fig. 3.1: Examples of some null gates discussed in Garcia-Escartin and Chamorro-Posada's "Equivalent Quantum Circuits" [7] . Being displayed are input/output cells from executing interactive Python within a Jupyter notebook using the Prove-It library with our quantum circuit extensions.

3. Equivalent Transformations. One process for optimizing a general algorithm for specific quantum hardware involves using a series of known quantum equivalences to transform the general algorithm so that it is optimized. Garcia-Escartin and Chamorro-Posada offer an extensive study of commonly used quantum equivalences [7]. They break down these equivalences into several categories: null gates, control reversal, deferred measurement, quantum classical substitution, and CNOT rearrangement.

Null gates handle the instance where a collection of gates have no effect on a qubit (Figure 3.1). This can be in reference to two gates that cancel each other out, such as two Hadamard gates, or any expression where the input is the same as the output such as a control gate where the control qubit is zero. These null gates can easily be replaced with either a blank wire or the Identity operator.

Control reversal refer to a situation where the control gate and the controlled gate can be reversed, either by themselves or with the help of additional gates (Figure 3.2). These reversals, like any other circuit equivalence can be verified with a matrix like Figure 3.2(b).



(a) A controlled Z gate can be easily reversed because either the control or the Z gate can be said to be the control.

	$\|00\rangle$	$\|01\rangle$	$\|10\rangle$	$\|11\rangle$
$\|00\rangle$	1	0	0	0
$\|01\rangle$	0	1	0	0
$\|10\rangle$	0	0	1	0
$\|11\rangle$	0	0	0	-1

(b) Evidence for this reversal is shown by the matrix operator above. If the order of the contents of each Ket (which represent the control input and the gate input) are reversed, we still end up with the same matrix.

Fig. 3.2: Z gate reversal and matrix evidence.

Deferred measurement takes into account the fact that a classically controlled gate that is controlled by a measurement gate is equivalent to a circuit where the measurement gate is moved farther along the wire and the classically control gate is replaced by a quantum control gate (Figure 3.3).

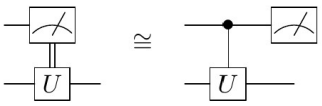


Fig. 3.3: A circuit with a measurement gate as a control is computationally equivalent to a circuit with a deferred measurement gate.

Quantum classical substitution relies on the principle that strictly classical operations are preferable to classically controlled quantum operations, which in turn are preferable to strictly quantum operations. This preference motivates the replacement of a controlled gate with a measurement gate or other equivalent classical options (Figure 3.4).

Lastly, CNOT gate rearrangement refers to various equivalences of circuit composed

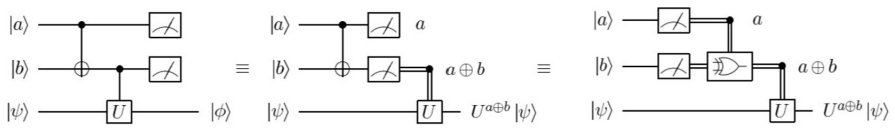


Fig. 3.4: Replacing quantum elements with equivalent classical elements is preferred to save quantum resources. Image from Garcia-Escartin and Chamorro-Posada's "Equivalent Quantum Circuits" [7].

solely of CNOT gates which are entirely analogous to classical circuit equivalences due to the linearity of CNOT and every other unitary operation (Figure 3.5).

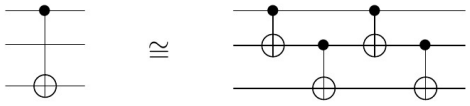


Fig. 3.5: An example of an equivalent CNOT rearrangement [7].

These equivalence rules allow us to replace portions of a larger circuit with an equivalent expression while maintaining the equivalence of the overall circuit. In this fashion, we can transform a general circuit to one that is optimized for specific quantum hardware. By using Prove-It to verify the validity of these transformations, we can verify that the entire optimized circuit is equivalent to the original circuit. Because we are only concerned with the portion of the circuit we are replacing, this process can be used for circuits of arbitrary size which allows us to generalize this process for *any* quantum circuit. Thus, the computation required for the verification of a quantum circuit equivalence has been linearized with respect to the size of the quantum circuit (e.g., number of quantum gate operations).

4. Circuit Implementation in Prove-It. By following the quantum equivalence framework outlined above, we can create theorems in Prove-It that define these equivalences. Afterwards, we can create additional theorems that allow equivalent pieces of a circuit to be swapped out with each other while still maintaining the overall equivalence of the circuit. With these theorem additions, we can concretely verify these transformations within circuits of an arbitrary size, thereby verifying that the resulting quantum circuit is equivalent to the first. In order to achieve this, several methods were added to Prove-It's theory of `physics.quantum` package to visually express quantum circuits, gates, literal circuit elements, wires, and multi-qubit gates. These methods allowed us to express any type of quantum circuit in Prove-It using commands listed in Table 4.1. In addition, the implementation of multi-qubit gates allowed us to create general theorems using Prove-It's ExprRanges that can be instantiated with circuits of arbitrary size (Figure 4.1).

A multi-qubit gate is formatted as a box around the label of the gate with vertical wires connecting each member of the multi-qubit gate. For example, in Figure 1.1 we see a specific multi-quibit gate containing a control and a target gate. However, a generic multi-qubit gate that uses Prove-It's ExprRange feature, is formatted differently than an explicit multi-qubit gate. When expressing a general multi-qubit gate acting on a non-specific set of

Circuit()	Defines a circuit element, takes in an ExprArray().
Gate(X)	Takes in a gate label and outputs a Gate element.
MultiQubitGate(X, Set(one, two))	Takes in a gate label and the positions of the other gates contained in the MultiQubitGate.
IdentityOp()	The identity operator. Produces a wire or an "I" gate depending on the indicated style.
MEAS	Produces a measurement gate.
SPACE	Produces an empty space.

Table 4.1: Common commands for creating a Circuit element in Prove-It.

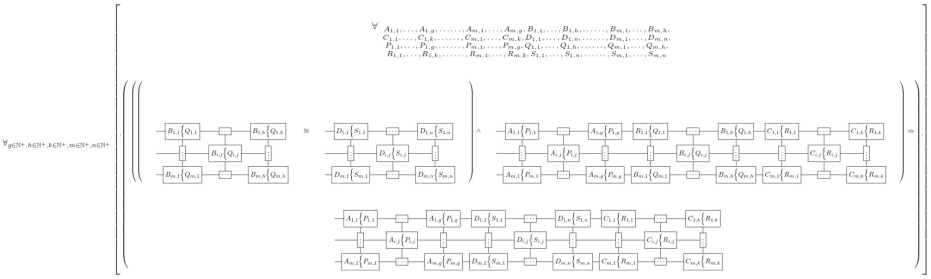


Fig. 4.1: Our temporal_circuit_substitution theorem that allows us to replace an equivalent portion of a valid (true) circuit of arbitrary size. The theorem states that for any valid quantum circuit, labeled in a manner that represents a true matrix equation for transforming input states to output states, the replacement of a portion of this circuit with an equivalent circuit of the same size results in another valid circuit. This is done with the use of generic multi-qubit gates and Prove-It's ExprRanges.

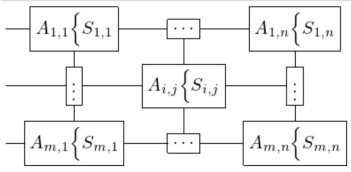
qubits, we display both a label for the gate as well as the expression that represents the set of qubits involved in the operation (Figure 4.2(a)). A multi-qubit gate acting on a specific set of qubits will only contain the label for the gate operation (Figure 4.2(b)). However, a multi-qubit gate acting on only one qubit *is* formatted with an explicit set in order to differentiate it from a gate (Figure 4.3). This is important for showing the reduction of a multi-qubit gate to a regular gate.

Using these multi-qubit gate properties, we have been able to create our temporal_circuit_substitution theorem (Figure 4.1) which will allow us to replace any portion of a valid (true) circuit with another equivalent circuit. A valid circuit is one that is labeled to express a quantum circuit that represents a true matrix equation for transforming the input quantum wavefunction to the output quantum wavefunction. As an example of a valid circuit,

$$|\psi\rangle \; -\boxed{U}- \; U\,|\psi\rangle \tag{4.1}$$

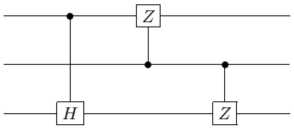
(under proper assumptions regarding ψ and U) represents the tautology $U\,|\psi\rangle = U\,|\psi\rangle$ (that

```
Circuit(ExprArray(
    ExprRange(i,
              ExprTuple(
                  ExprRange(j, MultiQubitGate(Aij, Sij), one, n)),
              one, m)))
```



(a) An example of a generic multi-qubit gate using Prove-It's ExprRange feature. Each box represents a gate. To the left of the large curly brace ($\{$) is the gate operation, indicated by the indexed A variable. To the right of the large curly brace is the set of qubits that the multi-qubit gate is acting on, indicated by the indexed S variable.

```
Circuit(
    ExprArray(
        ExprTuple(MultiQubitGate(CONTROL, Set(one, three)),
                  MultiQubitGate(Z, Set(one, two)),
                  IdentityOp(), IdentityOp()),
        ExprTuple(IdentityOp(), MultiQubitGate(CONTROL, Set(one, two)),
                  MultiQubitGate(CONTROL, Set(two, three)),
                  IdentityOp()),
        ExprTuple(MultiQubitGate(H, Set(one, three)), IdentityOp(),
                  MultiQubitGate(Z, Set(two, three)),
                  IdentityOp()))).withStyles(spacing='@C=3em @R=2em')
```



(b) When we have a specific collection of multi-qubit gates, we don't need to show the set of qubits that the multi-qubit gate is acting on because it is indicated by the vertical wires.

Fig. 4.2: Above we display both a generic multi-qubit gate (4.2(a)) and a specific multi-qubit gate (4.2(b)). Being displayed are input/output cells from executing interactive Python within a Jupyter notebook using the Prove-It library with our quantum circuit extensions.

```
# for use in reducing a MultiQubitGate to a gate within a Circuit.
unary_multiQubitGate_reduction = Forall(U,
                                        Forall(A,
                                               Equals(MultiQubitGate(U, Set(A)), Gate(U)),
                                               domain=NaturalsPos))
```

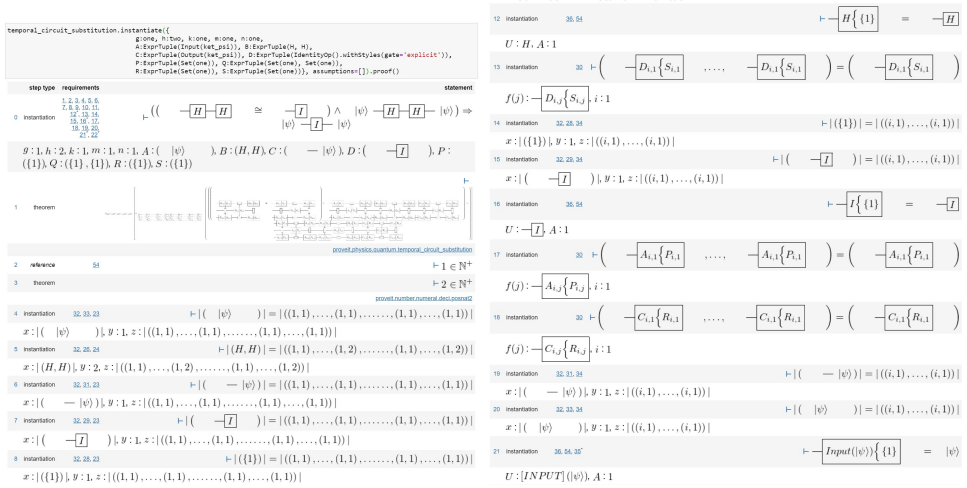
unary_multiQubitGate_reduction: $\forall_{U}\ \left[\forall_{A \in \mathbb{N}^+}\ \left(-\boxed{U\{\{A\}}- = -\boxed{U}-\right)\right]$

Fig. 4.3: Our "unary_reduction" theorem for reducing a multi-qubit gate that contains one gate to a single gate. This theorem is important for instantiating a general multi-qubit gate theorem (like Figure 4.1) with elements that are not multi-qubit gates. Analogous theorems for Inputs, Outputs, and the Identity Operator have also been created. Being displayed are input/output cells from executing interactive Python within a Jupyter notebook using the Prove-It library with our quantum circuit extensions.

is, U applied to the input $|\psi\rangle$ gives the output $U|\psi\rangle$). The temporal_circuit_substitution theorem is expressed using only multi-qubit gates but these will be reduced to single-qubit gates as appropriate. To do this, we created several "reduction theorems" that will take elements contained in a multi-qubit gate and reduce them to the actual element (Figure 4.3). For example, a multi-qubit gate that consists of only one gate can be reduced to a single gate. Similarly, an input Ket or output Ket that is contained in a multi-qubit gate can be reduced to just an input or output Ket. These reductions, among others, allow us to instantiate the generic temporal_circuit_substitution theorem with any circuit components in order to create a specific equivalent circuit. In Figure 4.4 we instantiate our temporal_circuit_substitution theorem to replace the null gates with the identity operator within a simple, valid circuit identity. Behind the scenes, Prove-It uses our reduction theorems to convert multi-qubit gates to single-qubit gates.

```
temporal_circuit_substitution.instantiate({
                                    g:one, h:two, k:one, m:one, n:one,
                                    A:ExprTuple(Input(ket_psi)), B:ExprTuple(H, H),
                                    C:ExprTuple(Output(ket_psi)), D:ExprTuple(IdentityOp().withStyles(gate='explicit')),
                                    P:ExprTuple(Set(one)), Q:ExprTuple(Set(one), Set(one)),
                                    R:ExprTuple(Set(one)), S:ExprTuple(Set(one))}, assumptions=[])
```

Fig. 4.4: The instantiation of our temporal_circuit_substitution theorem to replace the null gates with the identity operator. Being displayed are input/output cells from executing interactive Python within a Jupyter notebook using the Prove-It library with our quantum circuit extensions.



(a) The first eight steps of the proof for replacing a null gate with the identity operator. Steps 3-11 (9-11 are omitted for ease of readability) are internal theorems used to verify that the length of the ExprRange and the elements it is being replaced with are the same length.

(b) Steps 12-21 for the proof for replacing a null gate with the identity operator. Contrary to the first eight steps, the majority of these steps are internal theorems used for the reduction of multi-qubit gates to other circuit elements. Steps 12 and 14 are reducing a multi-qubit gate to a gate and steps 20 and 21 are reducing multi-qubit gates to the input and the output.

Fig. 4.5: The first 21 steps from Prove-It's proof output for Figure 4.4 (omitting 9-11 for ease of readability).

Figure 4.5 shows the first 21 steps of Prove-It's proof output for this particular instantiation. Steps 4-11, 14, 15, 19, and 20 are internal theorems used for verifying the length of the expressions being instantiated. Steps 12, 13, and 16-19, 21 are all reduction theorems used to transform a multi-qubit gate into the circuit element that is being instantiated. The full proof output is 58 steps. This is fairly long for a single instantiation, but Prove-It must satisfy quite a few internal requirements for this particular example. Each step is human-readable in convenient mathematical notation, and it was all handled via automation.

5. Conclusions. In the previous section we were able to demonstrate the circuit replacement capabilities of Prove-It with a simple null gate replacement. However, the theorems that were used are general enough to allow for the replacement of a sub-circuit within a larger circuit of arbitrary size. With this strategy, we can perform general quantum circuit transformations linear in time with respect to the number of expressed quantum operations because we are solving the guided quantum circuit equivalence problem rather than the general blind equivalence problem. With Prove-It's unique ability to represent expressions of arbitrary length using the ExprRange feature, we can represent circuits with arbitrary size, creating a powerful tool for visual quantum program verification.

6. Future Directions. In the future, we plan to create a more robust library of commands to customize the circuits further. Currently, users can create only the most basic elements of a circuit. More complex elements such as bell states, other types of measurement gates, or circuit labels have yet to be implemented. In addition, we plan to create many more theorems in order to truly verify the equivalence of a circuit transformation in Prove-It. While we were able to prove that the replacement of a portion of a circuit was valid, we did this by assuming that the two circuits were equivalent to begin with. Additional theorems will allow us to actually prove that the circuit being replaced is equivalent to the replacement. Furthermore, there are extensive formatting issues with the \QCircuit package. For example, circuits on either side of an operation (such as the $\wedge$ symbol in Figure 4.1) are formatted so that the top of the circuit is flush with the operation instead of being vertically centered with respect to the operation. As a result, many of the figures represented in this paper were edited with an external LaTeX editor in order to improve ease of readability. However, we plan to modify the \QCircuit package to improve the automatic formatting of the circuits. Furthermore, it may be useful to integrate an existing piece of quantum optimization software into Prove-It. This will allow users to automatically generate both an optimized circuit as well as a Prove-It proof certification to verify that the optimized circuit is equivalent to the original. Finally, we would like to extend Prove-It to support quantum algorithm verification using a variety of reasoning strategies that work together seamlessly in addition to the visual quantum circuit transformation technique we have discussed here. There are other diagramatic reasoning [5, 8], symbolic [15], and Feynmann path integral [1] approaches that could be incorporated.

7. Acknowledgements. We would like to thank Warren Craft from the University of New Mexico for his advice and help throughout this process. This work was supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research under the Quantum Computing Applications Team (QCAT) program.

REFERENCES

[1] M. Amy, *Towards large-scale functional verification of universal quantum circuits*, Electronic Proceedings in Theoretical Computer Science, 287 (2019), pp. 1–21.
[2] E. Ardeshir-Larijani, S. J. Gay, and R. Nagarajan, *Automated verification of quantum protocols by equivalence checking*, CoRR, abs/1312.5951 (2013).
[3] E. Ardeshir-Larijani, S. J. Gay, and R. Nagarajan, *Verification of concurrent quantum protocols by equivalence checking*, in International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Springer, 2014, pp. 500–514.
[4] A. Baltag and S. Smets, *Lqp: The dynamic logic of quantum information*, Mathematical Structures in Computer Science - MSCS, 16 (2006), pp. 491–525.
[5] R. Duncan and M. Lucas, *Verifying the steane code with quantomatic*, Electronic Proceedings in Theoretical Computer Science, 171 (2014), pp. 33–49.
[6] B. Eastin and S. T. Flammia, *Q-circuit tutorial*, 2004. arXiv:quant-ph/0406003v.
[7] J. C. Garcia-Escartin and P. Chamorro-Posada, *Equivalent quantum circuits*, 2011. arXiv:1110.2998 [quant-ph].

[8] L. Garvie and R. Duncan, *Verifying the smallest interesting colour codes with quantomatic*, arXiv:1706.02717, (2018).
[9] K. Hietala, R. Rand, S.-H. Hung, X. Wu, and M. Hicks, *A verified optimizer for quantum circuits*, 2019. arXiv:1912.02250 [cs.LO].
[10] T. Liu, Y. Li, S. Wang, M. Ying, and N. Zhan, *A theorem prover for quantum hoare logic and its applications*, 2016. arXiv:1601.03835 [cs.LO].
[11] D. Maslov, G. W. Dueck, D. M. Miller, and C. Negrevergne, *Quantum circuit simplification and level compaction*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 27 (2008), pp. 436–444.
[12] Y. Nam, N. J. Ross, Y. Su, A. M. Childs, and D. Maslov, *Automated optimization of large quantum circuits with continuous parameters*, npj Quantum Information, 4 (2018).
[13] D. Venturelli, M. B. Do, B. O'Gorman, J. Frank, E. Rieffel, K. E. C. Booth, T. N. Nguyen, P. Narayan, and S. Nanda, *Quantum circuit compilation : An emerging application for automated reasoning*. presented at ICAPS 2019 Workshop SPARK, 2019.
[14] W. M. Witzel, W. D. Craft, R. D. Carr, and J. E. M. Larrañaga, *Prove-it: A proof assistant for organizing and verifying general mathematical knowledge*, 2020. Available at https://github.com/PyProveIt/Prove-It/blob/master/ProveIt_Introduction.pdf.
[15] M. Ying and Z. Ji, *Symbolic verification of quantum circuits*, 2020. arXiv:2010.03032 [quant-ph].

EVALUATION OF ONEAPI FOR FPGAS

NICHOLAS MILLER*, JEANINE COOK†, AND CLAY HUGHES‡

Abstract. The high-performance computing (HPC) ecosystem increasingly supports heterogeneous architectures and customization. Field programmable gate arrays (FPGA) are among the options being considered due to their ability to both adapt to individual workloads and as prototype vehicles for application-specific accelerators. However, adoption has been limited due to the difficulty in programming these devices. To mitigate this, vendors are introducing frameworks based on embedded domain specific languages (eDSLs), such as SYCL. This work takes the first step in evaluating one of these new DSLs, DPC++, using DOE proxy applications to identify programmability gaps and performance on Intel FPGAs. Initial testing is being done with the MiniAMR application from the Mantevo suite, focusing on the 7-point stencil.

1. Introduction. As Moore's law comes to an end, we are seeing an increasing support for heterogeneous architectures that include domain- and application-specific accelerators. Application-specific accelerators (ASAs) are particularly appealing for many of the kernels of interest to the Advanced Simulation and Computing (ASC) Program, which supports the Department of Energy's (DOE's) National Nuclear Security Administration (NNSA) programs. These types of accelerators, which map applications directly to hardware, have already demonstrated promising performance and energy improvements for HPC kernels [1, 9, 5]. As such, next-generation specialized computing platforms could significantly accelerate mission applications compared to traditional central processing units (CPUs), general purpose accelerators like graphics processing units (GPUs), and even domain-specific accelerators like tensor processing units (TPUs). Unfortunately, ASAs have historically faced challenges for HPC, including a development environment not amenable to agile application and hardware co-design, system-integration and deployment complexity, and the fact that ASAs targeting a single application are ill-suited for platforms, like the NNSA's, that must support many diverse applications.

While ASAs built on reconfigurable field-programmable gate arrays (FPGAs) can address the latter challenge, only recently have HPC vendors embraced customization, which is reducing cost, fabrication, and integration challenges. Traditional HPC vendors are investing heavily in this area, embracing architectures like ARM and RISC-V that support customization, developing protocols for coherent accelerator and CPU communication (e.g., CXL, CCIX), as well as investing in new fabrication technologies like multi-chip modules that support physical integration of commodity and custom hardware. This renewed investment is driving innovations in hardware and software, as vendors recognize the need to provide both performance portability and languages extensions targeted at application developers, such as Intel's oneAPI [3] toolchain. However, the promise of performance portability and easy-to-use language extensions, with regard to FPGAs, remains unproven; system-level issues like data movement between a CPU and FPGAs must still be addressed and we do not yet know if FPGAs for Sandia would be effectively reconfigurable or reusable across Sandia's application domains. This research is the first step in answering these questions.

2. Background. In this section we will explore oneAPI and how it looks to create a programming model that both reduces the programmer's need to know digital logic design while also reducing the development time compared to traditional hardware description language (HDL) programming. Additionally an overview of the tools provided for programming using oneAPI along with a simple example of how data movement is handled is shown.

*University of Central Florida, nickmiller@knights.ucf.edu
†Sandia National Laboratories, jeacook@sandia.gov
‡Sandia National Laboratories, chughes@sandia.gov

Furthermore a discussion of related works is included.

2.1. oneAPI. oneAPI [3] is an open source programming model being developed by Intel with the intent of providing a single API that can be used on many different platforms, as shown in Figure 2.1. The core of oneAPI is a new programming language, DPC++, that builds upon the SYCL specification from The Khronos Group [7], which itself builds upon OpenCL [6], a standard used to describe hardware in a high level language, by providing a number of language extensions. oneAPI also includes many libraries that implement algorithms for applications such as video processing, data science, deep learning, and more.

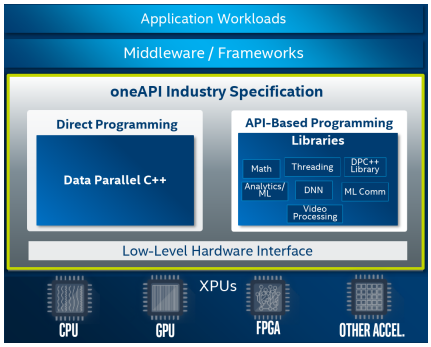


Fig. 2.1: oneAPI Layer Cake [3]

Historically, the traditional FPGA design workflow required knowledge of digital logic principles and hardware design; the oneAPI toolchain obviates the need for these skills. oneAPI looks to ease the burden of programming an FPGA through abstraction of the hardware, automating synchronization of data and associated operations, simplifying data transactions, and providing containers for more complex operations. Through these practices, oneAPI effectively removes the requirement of deep circuit-level knowledge and allows for a wider range of developers to have access to FPGA platforms. This is done by extending the SYCL interface with DPC++, which itself works as an abstraction layer for OpenCL. DPC++ handles the data transactions which would typically need to be done explicitly by the programmer in OpenCL while still allowing for the creation of kernels using OpenCL, albeit with a reduction in code complexity due to further abstraction of the kernel creation process. oneAPI further expands on SYCL by adding its own USM (unified shared memory) system used to provide pointer support on accelerator hardware and numerous extensions described in the oneAPI specification [3].

One flaw of oneAPI is the lack of documentation. Because it is still in beta, there is not a large library of examples to learn from. Additionally, the documentation can be disjointed between reading oneAPI, OpenCL, SYCL, and DPC++ documentation. This makes it difficult to get a full picture of how the program execution works and how to fully optimize code. To further compound the issue, there is a lack of information concerning exactly how the compiler picks the optimizations that it does and in-depth details of the hardware created by it.

2.2. Abstraction Upon Abstraction. DPC++ provides an abstraction layer to the hardware. By having this layer of abstraction, development time is greatly reduced compared to working with Verilog or VHDL. This layer of abstraction also alleviates the need to fully understand the hardware. For example, to create an accelerator in Verilog, one would typically need to create a PCIe interface that can send/receive PCIe packets, create a mechanism to parse the contents of those packets, manage the data coming to and from

the host, and then implement the algorithm. All of this is followed by a timing analysis and other performance enhancing optimizations to increase performance. Each of these steps is tedious and requires a deep understanding of hardware and standards, even with intellectual property (IP) blocks, illustrating the difficulty presented by the traditional FPGA development workflow. By using the oneAPI toolchain, most of these steps are abstracted away so that one only needs to write their algorithm in a C++-like language and compile. But there are some drawbacks to this level of abstraction.

The benefit to the traditional approach is that the programmer will have greater control of the resources being used and the ability to optimize it at a very low level in the design. Although the DPC++ compiler provides optimization passes, what they actually do at the register-transfer level (RTL) is often opaque and gives users very little direct control. Another downside of the DPC++ toolchain is the loss of the ability to use some of the FPGAs built in IP blocks, users are restricted to the interfaces described in the OpenCL Board Support Packages (BSP). However, these tradeoffs may be worthwhile given the learning curve and time needed to create an application. While the traditional approach can be done using block diagrams, there will still be a need to write in an HDL. This can be difficult as HDLs can take a long time to learn and become proficient in as they are unlike most high-level languages.

2.3. Data Movement and Placement. Modern architectures and programming languages require developers to handle data synchronization and structuring of program execution as to not cause issues with data dependencies. This can become a major time sink in development as keeping track of where the data is at all stages of execution is non-trivial. Additionally, handling how and when to transfer the data poses additional complexity. This is where oneAPI attempts to ease the development process through automation of these tasks. An example DPC++ code snippet is shown in Listing 1.

```
1  //Create the buffer and accessor to fill the buffer
2  sycl::buffer<int, 1> fpga_buffer(range<1>(bufferSize);
3  auto accessor = fpga_buffer.get_access<sycl::access::mode::read_write>(cgh);
4  //Filling it with example data
5  for(int i = 0; i < bufferSize; i++){
6      accessor[i] = i;
7  }
8  //Device queue submit
9  queue_event = device_queue.submit([&](sycl::handler& cgh) {
10     //The FPGA needs it's own accessor to the data
11     auto fpga_accessor = fpga_buffer.get_access<sycl::access::mode::read_write>(cgh);
12     cgh.single_task<class fpga_kernel>([=]() {
13         int sum = 0;
14         //Summing the data as an example computation
15         for(int i = 0; i < bufferSize; i++){
16                 sum += fpga_accessor[i];
17         }
18         //store the sum at the beginning of the array to reuse the buffer for writeback
19         fpga_accessor[0] = sum;
20     });
21 });
22 //Return the sum computed by the FPGA
23 return accessor[0];
```

Listing 1: Example Code Snippet

In this example, a chunk of data needs to move from the CPU to the FPGA and then back before execution can continue. Using DPC++, we can create a buffer on the host and send that data to the FPGA. Initially the CPU fills the buffer with the data it wants computed on by using an accessor to the buffer on line 6. This is done sequentially and will not call upon the device queue until the for loop is finished. Once the kernel is submitted to the device as seen at line 9-21 the execution of the program can continue asynchronously as

the CPU can continue execution while the FPGA completes its computations. Asynchronous execution, however, does stop at line 23 due to the buffer being accessed again. A blocking call to the accessor is made which cannot resolve until the FPGA is fully completed with its buffer accesses at lines 16 and 19. By using a buffer, the data does not need to be explicitly copied to and from the FPGA but instead is moved implicitly as each resource makes a request to use the data.

2.4. Tools. The central tool of the oneAPI toolchain is the DPC++ compiler itself. This is what most of oneAPI is built around as it is what creates the FPGA designs by leveraging the Quartus Prime software. This compiler uses LLVM/Clang as a base and adds the necessary components to efficiently interpret DPC++ into an OpenCL run time and FPGA design. The toolchain additionally provides the ability for users to perform emulation of a program, to check correctness of code. This is a very helpful capability as the compile times for this are much lower than that to create the full design. This allows for rapid development and testing without having to wait for the hours long compile times for the full design. However, this does have its faults as it cannot be used for profiling the expected performance of the program and the emulation may not always give the exact same result as hardware. For example, our test applications use float and double types; in emulation mode, the results are correct but when run on the FPGA floating-point division for both single and double precision were not IEEE754 compliant. The toolchain also creates reports based on the generated hardware design that shows the hardware usage, the pipelines created, and basic optimization data such as the latency for load and stores. This can be a useful tool when optimizing the code and finding pipeline stalls. Intel recommends using VTune for in-depth profiling. However in the early stages of this project it did not provide sufficient information to identify bottlenecks in program execution and was not used in this stage of the project but it is planned to be explored in future work. In place of VTune, CLILoader [2] was used to intercept OpenCL calls. This tool can show when kernels are queued, submitted, and executed by the OpenCL run time as shown in Figure 2.2. This tool provided insight into how the SYCL API calls were being interpreted by OpenCL run time calls and to see a timeline of their executions.

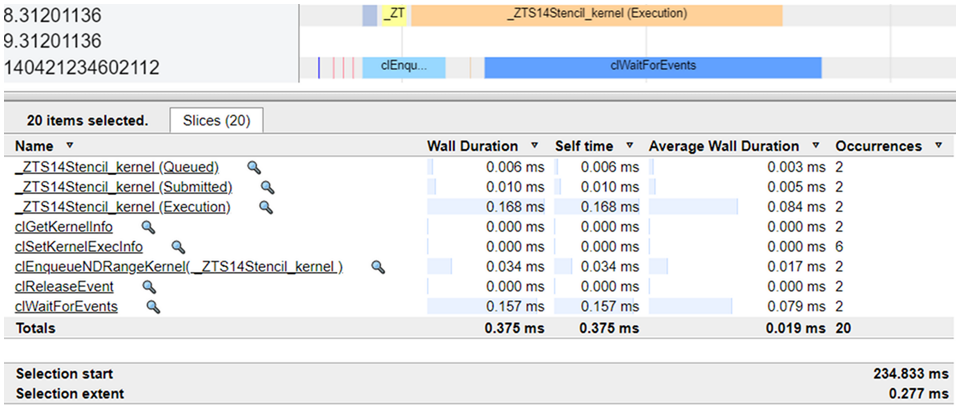


Fig. 2.2: CLILoader Output

2.5. Previous Work. Previous work has looked at the performance of OpenCL compared to CPU, GPU, and FPGA HDL devices. In [10] the authors translated a 3D fast fourier transform into an FPGA design using OpenCL. They found that while they could get better performance than CPU, GPU, or HDL designs they could only do so by not

using a full OpenCL created system but instead using it to create blocks, which were later integrated into their design. Further research such as [8] found that while an FPGA design was on average slower than a GPU in implementing a Smith-Waterman algorithm, it was still found to be competitive in performance with 1.56-3.78x better power efficiency. In [11] a discussion of how to optimize OpenCL for data throughput for a convolutional neural network is given compared to a handcrafted design.

Because SYCL is a relatively new standard, its performance has not been as thoroughly studied as OpenCL. SYCL becomes an attractive option for handling the data transactions and dependencies as described in [4] where it was found that the explicit data movements of OpenCL produced comparable performance results to those that were implicitly made by SYCL when using the same OpenCL kernel to do handwriting recognition.

3. Experimental Setup. For testing oneAPI for FPGAs, the Mantevo proxy application miniAMR [12] was chosen. miniAMR models an adaptive mesh refinement application that is usually run across multiple compute nodes. The code was created in a way to allow the calculation stage of the program to be exchanged with practically any grid computation that the programmer chooses but the default behavior is one that computes a 7-point average over the entire grid. This default case is used for all experiments. The program flow calculates the average in between communication of ghost values between nodes and fault tolerance checks. While this is not an ideal application for an FPGA to accelerate due to its low data reuse and high data dependency, its small code base, and straight-forward implementation does allow for the early understanding and programming using oneAPI.

The goal for this paper was to compare the performance of an Arria 10 FPGA development board (intel_a10g_pac) to a modern processor, an Intel Xeon Gold 6128 CPUs at 3.40GHz. All code was compiled using the oneAPI beta-08 dpcp++ compiler. To make the comparison, execution time was the metric selected to represent performance. To measure the execution time the built-in timer of miniAMR was used. Each experiment was run ten times and then averaged to get the runtime data. Slowdown was computed from the runtime data collected. The device utilization statistics came from the dpc++ compiler generated hardware reports. All experiments were run using the resources avilable on the Intel DevCloud.

Starting from a base conversion of the C-reference code of miniAMR into dpc++, experiments built upon one another. The first experiment combined the memory transactions between the host and the FPGA, which caused the area overhead on the FPGA to greatly increase which created the need to reduce the memory usage on the FPGA by reducing the amount of persistent data used in the kernel. Further performance increases were found by reducing the need to make memory conversions on the host side by storing all arrays as 1D arrays instead of 4D arrays due to the SYCL buffer needing the be 1D. Finally, buffering the kernel execution dependencies allowed for a more constant stream of kernel executions. After the code was optimized loop unrolling was done in order to observe the effects of having multiple kernel pipelines.

4. Results. This section discusses the modifications made to miniAMR and the impact that each of the optimization attempts had on performance. It should be noted that these optimizations are not intended to show the best way to optimize this application, as this study is still on-going, but to show how development proceeded. All comparisons are against the reference design running on the Xeon host processor with a single thread. In addition, AVX-512 extensions were evaluated for the reference design but it was found that the run time was slower than without them, so they are not included in the evaluation. In addition, the first port of the code is omitted in the discussion below because it had a slowdown of 249x relative to the reference implementation of miniAMR running on a Xeon. Code snippets for

all optimizations are provided in Appendix A. An overall slowdown comparison, relative to the Xeon, is shown in Figure 4.1 and resource utilization on the Arria 10 device is shown in Figure 4.2. The following subsections describe the cumulative optimizations for this code while the base code can be seen at Listing 4 and Listing 5.

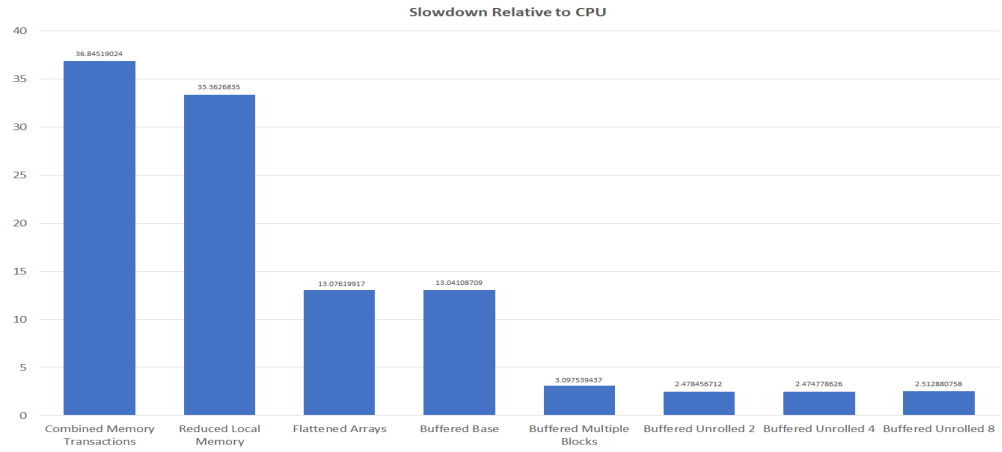


Fig. 4.1: Slowdown Relative to CPU

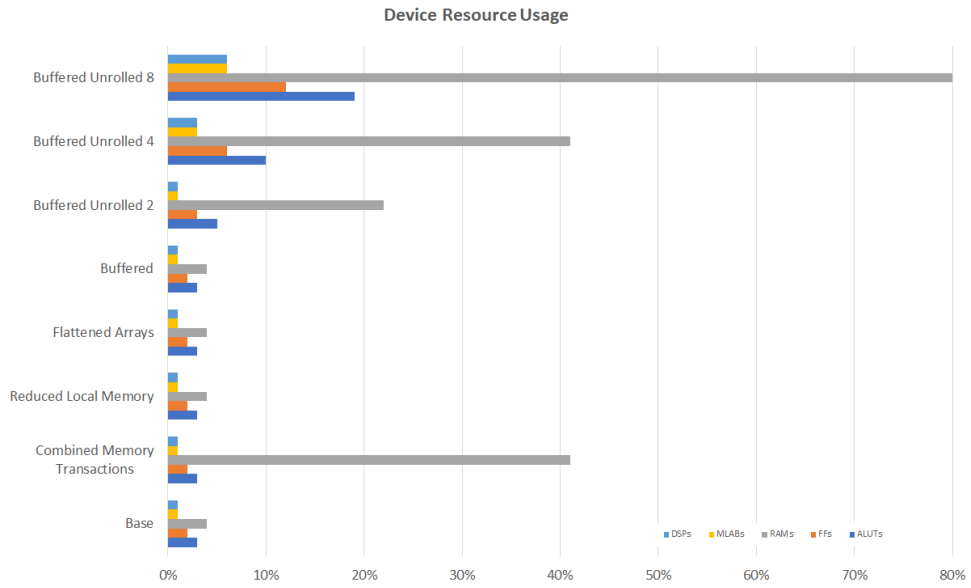


Fig. 4.2: FPGA Device Utilization

4.1. Combined Memory Transactions. The optimization that provided the largest performance boost was to combine all of the variable computations in a block into a single communication and computation step, as shown in Listing 6 and Listing 7 repectively.

By doing this, the number of calls to the SYCL runtime was reduced by 40x compared to the previous iteration of the code. These calls are used for communicating with the FPGA – submitting kernels to the command queue, waiting for responses, and transferring data. This resulted in a major run time decrease from the 249x slowdown of the base case down to 37x slowdown, as shown in Figure 4.1. This is primarily due to the fact that the previous execution time was limited by the latency of the SYCL runtime call latency. However, from Figure 4.2, it also increased the resource usage on the device, particularly the RAM usage because all 40 variables must be maintained in the local memory of the FPGA to complete the computations. This did not result in a commensurate increase the other resource utilizations as a single pipeline was still being used for the 7-point stencil.

4.2. Reduced Local Memory. This optimization was an attempt to reduce the explosion of BRAM consumption introduced in Section 4.1. The kernel modifications are shown in Listing 8. Instead of storing all variables on the FPGA at the same time, only the variable being currently worked on were stored in local BRAM cells. This reduced the BRAM utilization while keeping the rest of the device utilization relatively the same. This had a negligible affect on the overall run time, reducing the slowdown to 33x.

4.3. Flattening. This optimization attempts to further reduce the overhead of kernel and data management on the host side. The DPC++ programming language only supports 1D arrays inside of its buffers but, in miniAMR, the block data arrays are maintained as 4D arrays of [var][x][y][z]. This wreaks havoc with buffer creation because the 4D array must be converted to a 1D. This also results in additional storage overhead because a temporary array must be created for each kernel execution to hold the data going to and coming from the FPGA and complicates the algorithm because the data from the FPGA must be converted back to the 4D array and placed in the host memory, this results in a blocking call instead of relying on the buffer destructor, which triggers an automatic blocking update to the host memory. The modifications to how arrays are accessed in the entire code base allowed for a reduce amount of code for the setting up the data to be sent to the FPGA as can be seen in Listing 2. Due to the decreased memory movement and time spent transforming the array on each call, a drastic reduction in run time was seen, bringing the slowdown to 13x compared to reference with no increase in resource usage. This shows the importance of making the data structures for both the host and FPGA align properly as even with the speedup seen we have no increase in FPGA utilization.

4.4. Buffering. Buffering the communication and execution of the kernels can drastically improve performance when the kernel execution is long enough to hide the communication and SYCL runtime call latency. For our application, this is true as long as we are computing on a sufficient number of blocks at the same time. To set this up the program needs to be able to call the next kernel while the previous kernel is still executing. This is a fairly simple process as there is no longer a need to do any array conversion after the previous flattening optimization as can be seen in Listing 3. Buffer creation now uses a host pointer to the block data, which takes an insignificant amount of time compared to the kernel execution. For this optimization, there is no performance difference compared to the flattened experiment since they are both still operating on a single block. To show the difference that kernel buffering can make, another experiment, buffered_multiple_blocks, was run with an object passing through the grid. This causes multiple blocks to need to be computed upon. This test had up to 583 blocks at a single time, allowing for the buffering of each block between kernel executions and shows a considerable speedup at no increase of utilization as this is still a host side optimization. This optimization brings the slowdown to 2.4x that of the host CPU. Furthermore, the kernel code can be kept nearly the same as

shown in Listing 9.

4.5. Unrolling. The final optimization that we will discuss is an FPGA-side optimization of unrolling the outermost loop to create multiple pipelines that compute on variables independently. This showed a diminishing return in that after only creating 2 pipelines there was no speedup to be seen, while the device utilization increased. The cause for this is still under investigation but may be be due to reaching a cap on the bandwidth usage of the PCIe interface, the global memory system of the FPGA not being able to feed data to the pipelines fast enough, or data dependencies.

5. Future Work. The future work of this is to evaluate a larger number of proxy applications to find the functional usability of FPGA acceleration using the oneAPI toolchain. The approaches learned over the course of the miniAMR porting study will help inform the best approaches to take with other proxy applications. Additionally, a deeper understanding of the profiling tools, such as Intel's VTune, is needed in order to assure that the code is fully optimized to a production-ready state. While this paper did look at DPC++, a further evaluation of the libraries that are included in oneAPI is necessary to see how they may reduce the burden of programmers and the performance that they can provide. Furthermore a comparison of FPGA performance will be evaluated against both CPUs and GPUs for future proxy application development.

6. Conclusion. This paper presents an example for the development of a mini-app port to oneAPI. The programming model reduced the amount of time that it would take to produce a similar accelerator design using an HDL while producing results that showed a 2.4x slowdown compared to the reference design. For the fastest run time, the device utilization of the FPGA was still under 25% usage for BRAM and under 10% for all other resources, which would allow for additional algorithms to be implemented on the same design. While the design did not show a speedup compared to the reference, it still showed hopeful results as it was not considered to be an application that was expected to show speedup due to its data dependencies and low data reuse. The programming model for this design is considerably easier and required less of a background in digital logic design but still allowed for many optimizations to be tested.

7. Acknowledgements. The authors would like to thank Courtenay Vaughn and Simon Hammond from Sandia National Laboratories for their input on the miniAMR proxy application and the Center for Research into Novel Computing Hierarchies (CRNCH) at Georgia Tech for providing access to their Arria 10 nodes.

REFERENCES

[1] A. Abedalmuhdi, B. E. Wells, and K. Nishikawa. Efficient Particle-Grid Space Interpolation of an FPGA-Accelerated Particle-in-Cell Plasma Simulation. In *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 76–79, 2017.
[2] Intel Corporation. Intercept Layer for OpenCL Applications.
[3] Intel Corporation. oneAPI Specification, Release 0.7, April 2020.
[4] Anastasios Doumoulakis, Ronan Keryell, and Kenneth O'Brien. Sycl c++ and opencl interoperability experimentation with trisycl. In *Proceedings of the 5th International Workshop on OpenCL*, pages 1–8, 2017.
[5] Y. Gu and M. C. Herbordt. Fpga-based multigrid computation for molecular dynamics simulations. In *15th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2007)*, pages 117–126, 2007.
[6] Khronos OpenCL Working Group. OpenCL Specification, Version V2.2-11, July 2019.
[7] Khronos OpenCL Working Group. SYCL Specification: SYCL integrates OpenCL devices with modern C++, Version V1.2.1, Revision 6, November 2019.

[8] Enzo Rucci, Carlos Garcia, Guillermo Botella, Armando De Giusti, Marcelo Naiouf, and Manuel Prieto Matias. Accelerating smith-waterman alignment of long dna sequences with opencl on fpga. pages 500–511, 04 2017.

[9] A. Sanaullah, A. Khoshparvar, and M. C. Herbordt. FPGA-Accelerated Particle-Grid Mapping. In *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 192–195, 2016.

[10] Ahmed Sanaullah and Martin C Herbordt. Fpga hpc using opencl: Case study in 3d fft. In *Proceedings of the 9th International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies*, pages 1–6, 2018.

[11] Naveen Suda, Vikas Chandra, Ganesh Dasika, Abinash Mohanty, Yufei Ma, Sarma Vrudhula, Jae-sun Seo, and Yu Cao. Throughput-optimized opencl-based fpga accelerator for large-scale convolutional neural networks. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 16–25, 2016.

[12] C. T. Vaughan and R. F. Barrett. Enabling tractable exploration of the performance of adaptive mesh refinement. In *2015 IEEE International Conference on Cluster Computing*, pages 746–752, 2015.

Appendix A. Code Snippits.

In this appendix, we list key code snippits:

```
for (int in = 0; in < sorted_index[num_refine + 1]; in++) {
   bp = &blocks[sorted_list[in].n];
   sycl::range<1> num_array{ static_cast<size_t>(var_max *
      (x_block_size + 2) * (y_block_size + 2) * (z_block_size + 2)) };
   {
      sycl::buffer<double, 1> input_buffer(bp->array, num_array);
      fpga_kernel(input_buffer);
   }
}
```

Listing 2: Flattened Arrays Stencil

```
std::vector<sycl::buffer<double, 1>> input_buffer;

for (int in = 0; in < sorted_index[num_refine + 1]; in++) {
   bp = &blocks[sorted_list[in].n];
   input_buffer.push_back(sycl::buffer<double, 1>(bp->array,
      sycl::range<1>(static_cast<size_t>(var_max * (x_block_size + 2) *
      (y_block_size + 2) * (z_block_size + 2)))));
   fpga_kernel(input_buffer[in]);
}
```

Listing 3: Buffered Stencil

```
for (int in = 0; in < sorted_index[num_refine + 1]; in++) {
   bp = &blocks[sorted_list[in].n];
   for (var = 0; var < var_max; var++) {
      sycl::range<1> num_array{ static_cast<size_t>((x_block_size + 2) *
         (y_block_size + 2) * (z_block_size + 2)) };
      //create a buffer that goes to the fpga
      double* inputArray = new double[(x_block_size + 2) *
         (y_block_size + 2) * (z_block_size + 2)];
      //create a buffer that comes from the fpga
      double* outputArray = new double[(x_block_size + 2) *
         (y_block_size + 2) * (z_block_size + 2)];
      //flatten the 4d array to a 1d array for the buffer
      for (int i = 0; i <= x_block_size + 1; i++)
         for (int j = 0; j <= y_block_size + 1; j++)
            for (int k = 0; k <= z_block_size + 1; k++)
               inputArray[k + (z_block_size + 2) * (j + (y_block_size + 2) * i)]
                  = bp->array[var][i][j][k];
      sycl::buffer<double, 1> input_buffer(inputArray, num_array);
      {
         sycl::buffer<double, 1> output_buffer(outputArray, num_array);
         fpga_kernel(input_buffer, output_buffer);
      }//output_buffer detructor called here
      //write the data back to the block array
      for (int i = 1; i <= x_block_size; i++)
         for (int j = 1; j <= y_block_size; j++)
            for (int k = 1; k <= z_block_size; k++)
               bp->array[var][i][j][k] = outputArray[k + (z_block_size + 2) *
                  (j + (y_block_size + 2) * i)];
   }//input_buffer detructor called here
}
```

Listing 4: Base Stencil

```
void fpga_kernel(sycl::buffer<double, 1>& input_buffer,
sycl::buffer<double, 1>& output_buffer) {
   //Device queue submit
   queue_event = device_queue.submit([&](sycl::handler& cgh) {
      //Create FPGA side accessors to the buffers
      auto accessor_in =
         input_buffer.get_access<sycl::access::mode::read_write>(cgh);
      auto accessor_out =
         output_buffer.get_access<sycl::access::mode::discard_write>(cgh);
      cgh.single_task<class Stencil_kernel>([=]() {
         double work[12][12][12];
         double local_array[12][12][12];
         for (int i = 0; i <= 11; i++)
            for (int j = 0; j <= 11; j++)
               for (int k = 0; k <= 11; k++)
                  local_array[i][j][k] = accessor_in[i][j][k];
         for (int i = 1; i <= 10; i++)
            for (int j = 1; j <= 10; j++)
               for (int k = 1; k <= 10; k++)
                  work[i][j][k] = (
                     local_array[i - 1][j][k] +
                     local_array[i][j - 1][k] +
                     local_array[i][j][k - 1] +
                     local_array[i][j][k] +
                     local_array[i][j][k + 1] +
                     local_array[i][j + 1][k] +
                     local_array[i + 1][j][k]) / 7.0;
         for (int i = 1; i <= 10; i++)
           for (int j = 1; j <= 10; j++)
              for (int k = 1; k <= 10; k++)
                 accessor_out[i][j][k] = work[i][j][k];
   });
}
```

Listing 5: Base Kernel

```
for (int in = 0; in < sorted_index[num_refine + 1]; in++) {
   bp = &blocks[sorted_list[in].n];
   sycl::range<1> num_array{ static_cast<size_t>(var_max * (x_block_size + 2) *
      (y_block_size + 2) * (z_block_size + 2)) };
   //create a buffer that goes to the fpga
   double* inputArray = new double[var_max * (x_block_size + 2) *
      (y_block_size + 2) * (z_block_size + 2)];
   //create a buffer that comes from the fpga
   double* outputArray = new double[var_max * (x_block_size + 2) *
      (y_block_size + 2) * (z_block_size + 2)];
   //flatten the 4d array to a 1d array for the buffer
   for (var = 0; var < var_max; var++)
      for (int i = 0; i <= x_block_size + 1; i++)
         for (int j = 0; j <= y_block_size + 1; j++)
            for (int k = 0; k <= z_block_size + 1; k++)
               inputArray[(var * (x_block_size + 2) * (y_block_size + 2) *
                  (z_block_size + 2)) + (k + (z_block_size + 2) *
                  (j + (y_block_size + 2) * i))] = bp->array[var][i][j][k];
   sycl::buffer<double, 1> input_buffer(inputArray, num_array);
   {
      sycl::buffer<double, 1> output_buffer(outputArray, num_array);
      fpga_kernel(input_buffer, output_buffer);
   }
   //write the data back to the block array
   for (var = 0; var < var_max; var++)
      for (int i = 1; i <= x_block_size; i++)
         for (int j = 1; j <= y_block_size; j++)
            for (int k = 1; k <= z_block_size; k++)
               bp->array[var][i][j][k] = outputArray[(var * (x_block_size + 2) *
                  (y_block_size + 2) * (z_block_size + 2)) +
                  (k + (z_block_size + 2) * (j + (y_block_size + 2) * i))];
}
```

Listing 6: Combined Memory Transactions Stencil

```
1  void fpga_kernel(sycl::buffer<double, 1>& input_buffer,
2  sycl::buffer<double, 1>& output_buffer) {
3     //Device queue submit
4     queue_event = device_queue.submit([&](sycl::handler& cgh) {
5        //Create FPGA side accessors to the buffers
6        auto accessor_in =
7           input_buffer.get_access<sycl::access::mode::read_write>(cgh);
8        auto accessor_out =
9           output_buffer.get_access<sycl::access::mode::discard_write>(cgh);
10       cgh.single_task<class Stencil_kernel>([=]() {
11          //create a local copy of the array data for increased performance
12          double local_array[40][12][12][12];
13          for (int var = 0; var < 40; var++)
14             for (int i = 0; i <= 11; i++)
15                for (int j = 0; j <= 11; j++)
16                   for (int k = 0; k <= 11; k++)
17                      local_array[var][i][j][k] =
18                         accessor_in[(var * (12) * (12) * (12)) + (k + (12) *
19                            (j + (12) * i))];
20          for (int var = 0; var < 40; var++)
21             for (int i = 1; i <= 10; i++)
22                for (int j = 1; j <= 10; j++)
23                   for (int k = 1; k <= 10; k++)
24                      accessor_out[(var * (12) * (12) * (12)) + (k + (12) *
25                         (j + (12) * i))] = (
26                         local_array[var][i - 1][j][k] +
27                         local_array[var][i][j - 1][k] +
28                         local_array[var][i][j][k - 1] +
29                         local_array[var][i][j][k] +
30                         local_array[var][i][j][k + 1] +
31                         local_array[var][i][j + 1][k] +
32                         local_array[var][i + 1][j][k]) / 7.0;
33       });
34    });
35 }
```

Listing 7: Combined Memory Transactions Kernel

```
1  void fpga_kernel(sycl::buffer<double, 1>& input_buffer,
2  sycl::buffer<double, 1>& output_buffer) {
3     //Device queue submit
4     queue_event = device_queue.submit([&](sycl::handler& cgh) {
5        //Create FPGA side accessors to the buffers
6        auto accessor_in =
7           input_buffer.get_access<sycl::access::mode::read_write>(cgh);
8        auto accessor_out =
9           output_buffer.get_access<sycl::access::mode::discard_write>(cgh);
10       cgh.single_task<class Stencil_kernel>([=]() {
11          //create a local copy of the array data for increased performance
12          double local_array[12][12][12];
13          for (int var = 0; var < 40; var++)
14             for (int i = 0; i <= 11; i++)
15                for (int j = 0; j <= 11; j++)
16                   for (int k = 0; k <= 11; k++)
17                      local_array[i][j][k] =
18                         accessor_in[(var * (12) * (12) * (12)) + (k + (12) *
19                            (j + (12) * i))];
20             for (int i = 1; i <= 10; i++)
21                for (int j = 1; j <= 10; j++)
22                   for (int k = 1; k <= 10; k++)
23                      accessor_out[(var * (12) * (12) * (12)) + (k + (12) *
24                         (j + (12) * i))] = (
25                         local_array[i - 1][j][k] +
26                         local_array[i][j - 1][k] +
27                         local_array[i][j][k - 1] +
28                         local_array[i][j][k] +
29                         local_array[i][j][k + 1] +
30                         local_array[i][j + 1][k] +
31                         local_array[i + 1][j][k]) / 7.0;
32       });
33    });
34 }
```

Listing 8: Reduced Local Memory Kernel

```
1  void fpga_kernel(sycl::buffer<double, 1>& input_buffer) {
2     //Device queue submit
3     queue_event[kernelCounter % 2] = device_queue.submit([&](sycl::handler& cgh)
4     {
5        //Create accessors
6        auto accessor_in =
7           input_buffer.get_access<sycl::access::mode::read_write>(cgh);
8        cgh.single_task<class Stencil_kernel>([=]() {
9           double local_array[12][12][12];
10          #pragma unroll X //replace X with the number of unrolls 0, 2, 4, or 8
11          for (int var = 0; var < 40; var++) {
12             for (int i = 0; i <= 11; i++)
13                for (int j = 0; j <= 11; j++)
14                   for (int k = 0; k <= 11; k++)
15                      local_array[i][j][k] = accessor_in[(var * (12) * (12) *
16                         (12)) + (k + (12) * (j + (12) * i))];
17             for (int i = 1; i <= 10; i++)
18                for (int j = 1; j <= 10; j++)
19                   for (int k = 1; k <= 10; k++)
20                      accessor_in[(var * (12) * (12) * (12)) + (k + (12) *
21                         (j + (12) * i))] = (
22                         local_array[i - 1][j][k] +
23                         local_array[i][j - 1][k] +
24                         local_array[i][j][k - 1] +
25                         local_array[i][j][k] +
26                         local_array[i][j][k + 1] +
27                         local_array[i][j + 1][k] +
28                         local_array[i + 1][j][k]) / 7.0;
29          }
30       });
31    });
32 }
```

Listing 9: Buffered Kernel

III. Applications

Articles in this section discuss the application of computational techniques to simulate different physical systems.

1. *Cleaves and Wilson* used a global sensitivity analysis approach for the paramter selection and dimensionality reduction in **atomistic molecular dynamics (MD) simulations**, with focus on the energetic interactions between silicon and oxygen in the context of silica-based glasses. They reveal several non-influential parameters of the studied energetic properties.
2. *Ganesh, St. John, Lofstead, and Mitcheell* construct 3D simulations of **microstructure formation during metal additive manufacturing** for the metal selective laser sintering (SLS) process. In their method, they stitch together many smaller SPPARKS simulations on overlapping sub-volumes rather than simulating on the entire domain at once.
3. *Hanson, Bochev, and Paskaleva* utilized Dynamic Mode Decomposition (DMD), a system identification technique for learning reduced-order discrete-time dynamical systems from time series data based on singular value decomposition. Applied to **radiation-induced photocurrent in semiconductor devices**, they simulated the excess carrier density induced by radiation pulses by solving numerically the Ambipolar Diffusion Equation, then used the simulated internal state as training data for the DMD algorithm.
4. *Hothem and Parekh* study approximation schemes for k-local qubit systems and how they could be applied to **fermionic many-body systems with local interactions** to find the lowest energy or ground states.
5. *Krause and Steyer* explore Exponential Time Differencing Runge Kutta (ETD-RK) methods as an efficient time integrator for **the nonhydrostatic atmosphere model HOMME-NH**. They show how to use ETD-RK methods to maximize efficiency by keeping stage value computations and storage to a minimum, and compare the accuracy and stability regions of these methods.
6. *Machalek, Bran-Anleu, and Hecht* develop a **non-equilibrium model of a liquid hydrogen storage tank**. Their simulations included different boiling regimes with normal boiloff for model validation, and have highlighted key differences between an equilibrium and a non-equilibrium models.

A.A. Rushdi
M.L. Parks

November 1, 2020

GLOBAL SENSITIVITY DRIVEN INPUT DIMENSIONALITY REDUCTION FOR REAXFF PARAMETERIZATIONS OF SILICA-BASED GLASSES

HELEN CLEAVES* AND MARK WILSON†

Abstract. Accuracy of classical atomistic molecular dynamics (MD) simulations originates from the quality of the interatomic potential, defining pair-wise atomic energetic interactions. The reactive force field, ReaxFF is an example of a complex potential having multiple contributions to the system energy and therefore multiple parameters to define the potential. In an effort to understand the relationships between these parameters and simulated properties, we deploy a global sensitivity analysis approach to screen these parameters for importance and potential dimensionality reduction. We focus our analysis on the energetic interactions between silicon and oxygen in the context of silica-based glasses. A set of numerical results is presented which reveal several non-influential parameters of the studied energetic properties for this system.

1. Introduction. Molecular dynamics (MD) simulations with reactive potentials provide the capability to gain chemically specific, atomic-scale insight into a broad range of material science applications with examples including dislocation dynamics, chemical kinetic processes, and crack propagation. Each of these examples represent a complex dynamic process where MD can expose the atomic-scale origins of the given phenomena, which can prove useful for designing preventative or predictive measures.

ReaxFF is a reactive interatomic potential [8, 1, 10] which can be utilized by the MD simulation package LAMMPS [6]. Given an accurate parameterization, LAMMPS paired with ReaxFF is capable of computing highly specific chemical data. Every unique material system and physics of interest requires a corresponding unique parameterization. There are 72 parameters involved in a reactive potential parameterization for a single element, and for compounds involving multiple distinct elements the parameter dimension is significantly larger. Due in part to the shear amount parameters that need to be estimated, creating novel parameterizations is a challenging and time consuming task. To make ReaxFF more extensible for projects that require fast analysis of novel (multi-component) material systems, the goal of the present work is to use global sensitivity analysis (GSA) to identify non-influential parameters to the ReaxFF parameterization.

We focus our analysis on a well-utilized, existing parameterization for the elements silicon (Si) and oxygen (O) in the context of silica-based glasses [5]. We study a variety of quantities of interest (QoIs) corresponding to the energetic interactions between Si and O, and screen for non-influential parameters by computing derivative-based global sensitivity measures (DGSMs) [4, 2] for the associated input parameters. Studied QoIs include isolated (single) atom energy, radial bond energy, angular bending energy, and torsional rotation energy.

The article is organized as follows: In Section 2 we explain the uncertain parameter space and give a brief overview of the studied objectives. A concise overview of DGSMs definitions and properties for both scalar and function-valued QoI is included in Section 3. We provide a set of numerical results and explanations in Section 4. Lastly, in Section 5 we include possibilities for future work and closing remarks.

2. ReaxFF parameters and QoIs. Here we provide a brief overview of how ReaxFF works, the parameters associated with a given compound, and expand on the QoIs to be studied.

*North Carolina State University, hlcleave@ncsu.edu
†Sandia National Laboratories, marwils@sandia.gov

Parameters of ReaxFF ReaxFF is a bond order-based interatomic potential, which means it treats bond order as a continuous function of interatomic distance. Due in part to these factors, ReaxFF is able to compute atomistic-scale chemical behavior. ReaxFF is known for its near-first principles accuracy. For brevity we do not include the precise analytical equations in the present work. A detailed breakdown of the analytical form of the involved potentials can be accessed in reference [9].

There are six different types of parameters in the ReaxFF potentials: atom, bond, off-diagonal, angle, torsion, and hydrogen bond. There are a handful of parameters which are fixed for certain atoms or compounds. For example, there is a parameter corresponding to atomic mass which is fixed for every atom. In Table 4.1 we include the parameter types along with the total number of parameters associated with each QoI.

The input parameters are assumed to be independent, and uniformly distributed such that $\theta_j \sim \mathcal{U}(a_j, b_j)$ where a_j and b_j are the lower and upper bounds corresponding to parameter θ_j. The upper and lower bound for each parameter were determined by numerical experimentation. The interval for each θ_j was established around the published values defined in the work of Pitman et al [5]. Then, the QoI functions were evaluated with parameters sampled from these intervals. Upper and lower bounds were modified until regions were found for which LAMMPS and ReaxFF did not error. The intention was to test the extent at which the parameter space is limited by the calculation done by LAMMPS and ReaxFF, rather then by the physical meaning of the parameters.

QoI overview We consider four QoIs: isolated (single) atom energy, radial bond energy, angular bending energy, and the torsional rotation energy. We denote these QoI as $S_i(\boldsymbol{\theta})$, $B_{ij}(x, \boldsymbol{\theta})$, $A_{ijk}(x, \boldsymbol{\theta})$, and $T_{ijkl}(x, \boldsymbol{\theta})$ respectively. The subscripts correspond to the atom numbers. Note, the single atom energy is a function of the input parameters $\boldsymbol{\theta}$ only, while the remaining three QoIs are also functions of an independent state variable x.

We briefly describe the LAMMPS simulation for each QoI studied. For $S_i(\boldsymbol{\theta})$ a single atom is placed in a "box", and LAMMPS calculates the associated potential energy. For $B_{ij}(x, \boldsymbol{\theta})$, two atoms are placed 1 Angstrom apart and then separated to a distance of 14.9 Angstrom. LAMMPS calculates the potential energy of the bond at intervals of 0.1 Angstrom. For this QoI the state space x represents distance. The simulation for $A_{ijk}(x, \boldsymbol{\theta})$ involves three atoms. The atoms are initialized with an angle of 10 degrees between them. Then, a radial atom is moved until the angle between all three atoms is equal to 180 degrees. LAMMPS measures the angular energy at ten degree intervals. In this case, the state space corresponds to degrees. Lastly, the simulation for $T_{ijkl}(x, \boldsymbol{\theta})$ involves rotation the bond between atoms j and k a full 360 degrees. LAMMPS measures torsion rotational energy at intervals of one degree. A visual representations for each QoI are displayed in Figure 2.1.

In the present approach we assume a QoI depends on all possible relevant parameters. For example, we assume B_{ij} depends on all possible atom and bond and off-diagonal parameters. Therefore, to simulate the bond potential energy of Si-O we would need two sets of atom parameters (one for Si and one for O), as well as a set of bond parameters, and a set of off-diagonal parameters. This QoI may therefore require as many as 86 parameters. Assuming very little about which input parameters are relevant to a QoI allows us to develop a framework that can be applied to a wide range of QoIs. For the QoIs studied we will see that not all included parameters have an effect on their respective QoIs, see: Section 4.

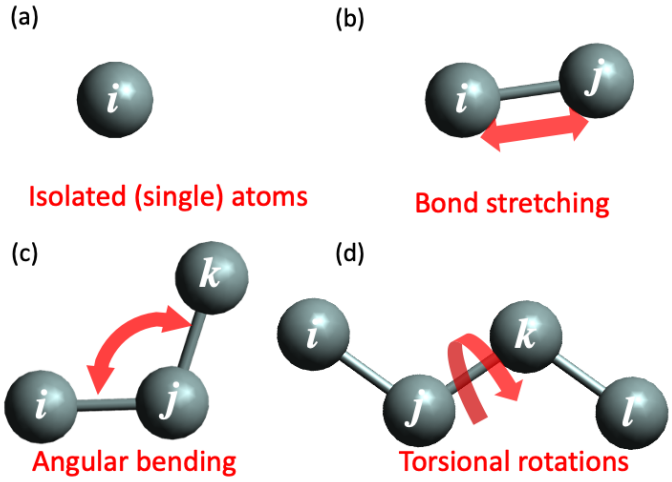


Fig. 2.1: Visual representations of the four studied QoIs. Clockwise from upper left: S_i, B_{ij}, A_{ijk}, and T_{ijkl}.

3. Methods. In this section, we supply a concise overview of both scalar and function-valued DGSMs. Let Ω be a $\mathrm{N_p}$-dimensional probability space with probability measure μ. Let $f(s, \boldsymbol{\theta})$ be a measurable function from $\mathcal{X} \times \Omega \to \mathbb{R}$, where $\boldsymbol{\theta}$ is the $\mathrm{N_p}$-dimensional vector of uncertain parameters, and $s \in \mathcal{X} \subseteq \mathbb{R}$ is an independent state variable, representing either space or time. We assume $\frac{\partial f}{\partial \theta_i}$ is square integrable in the product space for all $i = 1, \ldots, \mathrm{N_p}$. Observe, for a fixed $\hat{s} \in \mathcal{X}$, $f(\hat{s}, \cdot)$ is a scalar value. We have the following definition of the scalar derivative-based global sensitivity measure (DGSM) for f with respect to the uncertain parameter θ_j:

$$\nu_j(f(\hat{s}, \cdot)) := \int_\Omega \left(\frac{\partial f(\hat{s}, \boldsymbol{\theta})}{\partial \theta_j} \right)^2 \mu(d\boldsymbol{\theta}). \tag{3.1}$$

The above definition was developed in references [4, 3] for scalar QoI and can be interpreted as follows: a parameter θ_j with a relatively "small" ν_j suggests that f is not very sensitive to changes in that parameter. DGSMs are commonly used to screen for unimportant parameters by picking some "importance" threshold, and assuming all parameter with DGSM values below this threshold to be non-influential. Identifying non-influential parameters has a variety of applications, including input dimension reduction and increased understanding of QoI behavior. The isolated single atom energy $S_i(\boldsymbol{\theta})$ is an example of a scalar QoI.

Direct computation of the DGSMs requires estimating an integral over the uncertain parameter space. This is often computed via Monte Carlo or Latin hypercube sampling, a procedure that becomes computationally expensive as the input parameter dimension increases. However, it has been observed in practice that the number of samples required for a sufficient accurate estimate of ν_j is not too large [4]. We must also approximate partial derivates with respect to input parameters which can be accomplished in a variety of ways. The approach taken in the present work is finite differences.

Excluding the isolated single atom energy, all studied QoIs are function-valued. This means they are functions of both the parameters space as well an independent state variable. A generalization of the scalar DGSMs is need for such QoIs. This is accomplished by

computing the scalar DGSM value at $s \in \mathcal{X}$ and then averaging over the state space. This approach is purposed in reference [2] resulting in the following definition for function-valued DGSMs:

$$N_j(f) := \int_{\mathcal{X}} \nu_j(f(s,\cdot)\, ds \tag{3.2}$$

The interpretation of function-valued DGSMs is the same as scalar DGSMs, i.e. a smaller value implies a less important parameter. We compute ν_j the same as in the scalar case, and the integral over the state space is accomplished using quadrature.

Lastly, we mention that the motivation for using DGSMs is based primarily on a desire to build in future flexibility for restrictions on the input parameters. Recall, we have assumed the input parameters are all independent. This assumption may not hold for every future QoI studied. Unlike some other sensitivity analysis metrics, such as Sobol' indices [7], which rely on input parameter independence, DGSMs do not require any such assumption to be applicable.

4. Numerical Results. We let the number 1 represent an oxygen atom, and let 2 represent a silicon atom. We compute the DGSM values for 12 QoIs. Parameters with nonzero DGSM values are considered influential. For all DGSM computations we use finite differences to approximate the partial derivatives, and for the 10 function-valued QoIs we use quadrature to compute the integral over the state space. A summary of the studied QoIs, total number of parameters, influential parameter numbers, and total amount of QoI samples used is provided in Table 4.1.

Isolated (Single) Atom Energy Recall, $S_j(\boldsymbol{\theta})$, $j = 1, 2$ is a scalar QoI, corresponding to the isolated (single) energy of an atom. We fix six of the atom parameters and assume $S_j(\boldsymbol{\theta})$ is a function of the remaining $\mathrm{N_p} = 26$ atom parameters. Using a sample size of $\mathrm{N_s} = 1{,}000$ we compute the DGSMs for both atom 1 and atom 2, respectively. Observe that from the original 26 parameters, only *two* parameters have nonzero DGSM values. This suggests that the single atom for both O and Si depends on the same two parameters, only.

Radial Bond Energy Next, we compute the function-valued DGSMs for the radial bond energy $B_{ij}(x, \boldsymbol{\theta})$. For $i = j = 1, 2$ the radial bond energy is assumed to be a function of $\mathrm{N_p} = 42$ atom and bond parameters. There are no off-diagonal parameters when $i = j$. We compute the function-valued DGSM values for B_{11} and B_{22} with $\mathrm{N_s} = 1{,}000$. The QoI B_{11} has 12 parameters with zero DGSM values and B_{22} has 17 parameters with zero DGSM values. Therefore, the input parameter dimension of B_{11} and B_{22} can be reduced to 30 and 25, respectively. Note, there were several parameters that were influential for B_{11} that were non-influential for B_{22}. This could be due in part to the differences in the atomic structure of the two different atoms, and the associated bonding behaviors.

We also compute the function-valued DGSMs for B_{12} with $\mathrm{N_s} = 1{,}000$. This QoI is assumed to be a function of two sets of atom parameters, one set of bond parameters, and one set of off-diagonal parameters. After excluding fixed parameters the input parameter dimension for B_{12} is $\mathrm{N_p} = 73$. There are 33 parameters with zero DGSM values reducing the input dimension from 73 to 40.

Angular Bending Energy Recall, $A_{ijk}(x, \boldsymbol{\theta})$ represents the angular bending energy. For the case when $i = j = k = 1$ A_{111} is a function of $\mathrm{N_p} = 56$ atom, bond, angle, and torsion parameters. For $i = j = k = 2$ A_{222} is a function of $\mathrm{N_p} = 49$ atom, bond, and angle parameters. We compute the DGSM values for A_{111} with $\mathrm{N_s} = 1{,}000$ and determine there are 41 influential parameters. Similarly, we use $\mathrm{N_s} = 991$ to compute the DGSM values for A_{222}. This QoI had 37 influential parameters. The 4 parameter difference between A_{111} and A_{222} could once again be a consequence of the atomic structure of the two different atoms, and the associated bonding behaviors.

Table 4.1: List of studies performed in this work. Each quantity of interest (QoI), isolated (single) atom energy ($S_i(\theta)$) radial bond energy ($B_{ij}(x,\theta)$), angular bending energy ($A_{ijk}(x,\theta)$), torsional rotational energy ($T_{ijkl}(x,\theta)$) is listed with their respective varied parameters, total number of samples (N_s), and number of samples with LAMMPS errors (N_e). Subscripts denote atom types, where 1 represents oxygen and 2 silicon.

QoI	parameters	N_p	influential parameters	N_s	N_e
$S_1(\theta)$	atom: 1	26	2	1000	0
$S_2(\theta)$	atom: 2	26	2	1000	0
$B_{11}(x,\theta)$	atom: 1 bond: 1_1, 2_2	42	30	1000	0
$B_{22}(x,\theta)$	atom: 2 bond: 2_2	42	25	1000	0
$B_{12}(x,\theta)$	atom: 1,2 bond: 1_2, off-diagonal: 1_ 2	73	40	1000	0
$A_{111}(x,\theta)$	atom: 1 bond: 1_1 angle: 1_1_1 torsion: 1_1_1_1	56	41	1000	0
$A_{222}(x,\theta)$	atom: 2 bond: 2_2 angle: 2_2_2	49	37	1000	9
$A_{122}(x,\theta)$	atom: 1,2 bond: 1_1, 1_2 off-diagonal: 1_2 angle: 1_2_2, 2_1_2	103	75	1000	26
$A_{212}(x,\theta)$	atom: 1, 2 bond : 1_2, 2_2 off-diagonal: 1_ 2 angle: 1_2_2, 2_1_2	103	76	1000	20
$A_{121}(x,\theta)$	atom: 1, 2 bond : 1_1, 1_2 off-diagonal: 1_ 2 angle: 1_2_1, 1_1_2 torsion: 1_1_1_1	110	79	1000	12
$A_{112}(x,\theta)$	atom: 1, 2 bond : 1_1, 1_2 off-diagonal: 1_ 2 angle: 1_2_1, 1_1_2 torsion: 1_1_1_1	110	78	1000	16
$T_{1111}(x,\theta)$	atom; 1 bond : 1_1 angle: 1_1_1 torsion: 1_1_1_1	56	46	1000	612

The QoIs A_{122} and A_{212} are functions of $N_p = 103$ atom, bond, off-diagonal, and angle, parameters. We use $N_s = 974$ and $N_s = 980$ samples to compute the DGSM values for A_{122} and A_{212} respectively. There are 75 influential parameters for A_{122} and 76 influential parameters for A_{212}. In this instance, the QoIs shared all the same non-influential parameters except one parameter corresponding to bond over-coordination energy. This difference could be a consequence of the structure of the compound.

For the QoIs A_{121} and A_{112} we have $N_p = 110$ parameters. We compute the DGSM values with $N_s = 988$ and $N_s = 984$ samples for A_{121} and A_{112}, respectively. The results are used to determine there are 79 influential parameters for A_{121} and 78 influential parameters

for A_{112}. Similar to the previous mixed atom angular bending energy, this difference may be a consequence of the atom structure.

Torsional Rotational Energy Lastly, we consider the torsional rotational energy for T_{1111} which is a function of 56 input parameters. We compute the DGSM values using $\mathrm{N_s} = 384$ samples. Note this is significantly smaller sample size. There were 10 parameters with zero DGSM values. Therefore input parameter dimension can be reduced from 56 to 46.

5. Conclusions. A global sensitivity approach was utilized to identify non-influential parameters of the reactive potentials used by ReaxFF. We focused on energetic interactions between Si and O, and calculated DGSM values for single atom energy, radial bond energy, angle bending energy, and torsional rotation energy, for a variety of atom combinations. The computed sensitivity measures indicated that several of input parameters are insignificant in this material system. This information could be used to inform the user of parameters that are non-influential during the optimization process for Si-based glass ReaxFF input parameters.

Future work is needed. With respect to the parameter space, we made several simplifying assumptions, including parameter independence. Also, the parameter intervals were determined experimentally. Future efforts to better map out the parameter space and relationships between parameters is needed. Future work could also include comparing results for elements in similar periodic categories and compounds with similar structures in order to look for trends.

We observed a zero DGSM value for many parameters. In future studies, an importance threshold could be enforced on the DGSM values to reduce the input dimension to a size for which more informative, but computationally expensive GSA tools can be applied. Once the input dimension has been reduced, the accuracy of the reduced simulations could be evaluated by comparing to simulations computed in the full parameters space. Additionally, input parameter dimension reduction could in turn inform which parameters need precise estimation for different QoIs to achieve an accurate parameterization.

REFERENCES

[1] K. Chenoweth, A. C. Van Duin, and W. A. Goddard, *Reaxff reactive force field for molecular dynamics simulations of hydrocarbon oxidation*, The Journal of Physical Chemistry A, 112 (2008), pp. 1040–1053.

[2] H. L. Cleaves, A. Alexanderian, H. Guy, R. C. Smith, and M. Yu, *Derivative-based global sensitivity analysis for models with high-dimensional inputs and functional outputs*, SIAM Journal on Scientific Computing, 41 (2019), pp. A3524–A3551.

[3] S. Kucherenko et al., *Derivative based global sensitivity measures and their link with global sensitivity indices*, Mathematics and Computers in Simulation, 79 (2009), pp. 3009–3017.

[4] S. Kucherenko, M. Rodriguez-Fernandez, C. Pantelides, and N. Shah, *Monte carlo evaluation of derivative-based global sensitivity measures*, Reliability Engineering and System Safety, 94 (2009), pp. 1135 – 1148.

[5] M. C. Pitman and A. C. Van Duin, *Dynamics of confined reactive water in smectite clay–zeolite composites*, Journal of the American Chemical Society, 134 (2012), pp. 3042–3053.

[6] S. Plimpton, *Fast parallel algorithms for short-range molecular dynamics*, Journal of computational physics, 117 (1995), pp. 1–19.

[7] I. Sobol, *Global sensitivity indices for nonlinear mathematical models and their monte carlo estimates*, Mathematics and Computers in Simulation, 55 (2001), pp. 271–280.

[8] A. C. Van Duin, S. Dasgupta, F. Lorant, and W. A. Goddard, *Reaxff: a reactive force field for hydrocarbons*, The Journal of Physical Chemistry A, 105 (2001), pp. 9396–9409.

[9] A. C. Van Duin, S. Dasgupta, F. Lorant, and W. A. Goddard, *ReaxFF: A reactive force field for hydrocarbons*, Journal of Physical Chemistry A, 105 (2001), pp. 9396–9409.

[10] A. C. Van Duin, W. Goddard, M. Islam, H. Van Schoot, T. Trnka, and A. Yakovlev, *Reaxff 2019.3, scm.*

STITCH-CAD: CREATING DIGITAL TWIN SPPARKS SIMULATION FOR ADDITIVE MANUFACTURING

MANISHA GANESH*, GAVIN ST. JOHN†, JAY LOFSTEAD‡, AND JOHN MITCHELL§

Abstract. The objective of this project is to construct 3D simulations of micro-structure formation during metal additive manufacturing for the metal selective laser sintering (SLS) process. SPPARKS is an existing kinetic Monte Carlo simulation framework for simulating micro-structure evolution on a 3D domain. Using traditional methods for computing and writing to files is prohibitive because of length scale disparities between engineered component and micro-structures. *Stitch-IO* helps overcome this by allowing the user to *stitch* together many smaller SPPARKS simulations on overlapping sub-volumes rather than simulating on the entire domain at once; simulation results are appended and stitched together in one database file. *Slic3r* is open source software typically used to generate the G-code for 3D printers given an input CAD file representing component geometry. *Stitch-cad* combines these three systems enabling a user to simulate and visualize micro-structure evolution of a part, layer by layer, as it is printed; the completed simulation data is stored in one file which may be visualized and analyzed on any arbitrary sub-volume using standard tools or through a simple Python or C API. Initial limited comparative testing has shown results consistent with physical builds.

1. Introduction. Offering a digital twin enables computer modeling of systems without have to physically build and test prototypes thereby potentially speeding development and saving time and money. The combination of SPPARKS [3] and Stitch-IO [1] has previously been demonstrated as enabling large, length scale simulations of micro-structures arising from metal Additive Manufacturing (AM). However, the models used were simple and directly programmed in SPPARKS. This new Stitch-CAD effort enhances the existing model by using geometric information contained in an input CAD model to steer SPPARKS via G-code in the same way an AM machine operates to manufacture a metal part. In this way, Stitch-CAD creates a digital twin of the manufactured part using the same geometric process parameters used by the AM machine. *Stitch-CAD* is focused on simulation of metal selective laser sintering (SLS) 3D printing.

Additive manufacturing (AM) refers to the process in which material is incrementally deposited and merged (rather than subtracted), typically layer by layer, in order to construct a three-dimensional object. The geometry of these objects are defined by CAD models, often represented in the STL file format. In order to print the object, the STL file is fed into a slicing tool that splits the 3D object into layers and generates a set of layerwise print instructions. These instructions are written as G-code which is the industry standard for driving AM machines.

This paper shows initial outcomes from using Stitch-CAD to explore model creation. These tests demonstrate the simulation can be used to tune physical build parameters reducing the number of faulty builds due to poor parameter choices.

The paper is organized as follows. First are short sections summarizing SPPARKS and Stitch-IO in Sections 2 and 3, respectively. Section 4 is next which details the new Stitch-CAD mechanisms and procedures. The new algorithm is presented and described in Section 5. Our main results are presented in Section 6. Conclusions follow in Section 7.

2. SPPARKS Overview. Welding and additive manufacturing (AM) models [6] have been successfully incorporated into the *spparks* [2, 4] kinetic Monte Carlo framework. For the purpose of engineering design and analysis, material microstructures are predicted by

*University of Texas at Austin, manishaganesh@utexas.edu
†University of Florida, gavinastjohn@gmail.com
‡Sandia National Laboratories, gflofst@sandia.gov
§Sandia National Laboratories, jamitch@sandia.gov

simulating the process of melt, fusion and solidification. In these models, a heat source, typically a laser, creates a very localized melt pool and surrounding region called the heat affected zone (HAZ); outside of the HAZ, the temperature is below the threshold for grain growth and evolution. In the case of welding, the laser heat source moves along a joint melting and fusing material from both sides of the joint which subsequently solidifies. At any particular instance during the process, models need only consider the localized HAZ around the laser location. Material in the remaining areas outside of the HAZ remains unchanged.

3. Stitch-IO Overview. Stitch-IO is a new IO library that enables the progressive simulation model by breaking some fundamental assumptions other IO libraries require. First, the simulation domain is never formally defined. Instead, the formal simulation domain is inferred based on the spatial locations provided as part of each output operation. Second, the entire simulation domain for a given timestep is never stored during any particular simulation. Instead, only a portion of domain is in memory during a simulation. When the state for a given region of the domain is requested, the correct state is constructed based on the provided time epoch. Finally, the library is fully Python enabled and uses a standard embedded database format internally enabling both ad hoc querying using Python primitives for quick visualizations and data analysis and direct data storage access, if needed, using standard APIs. This latter openness enables users to extend Stitch-IO functionality without having to change the library. Instead, arbitrary operations on stored data can be performed. The database format also offers concurrency control, resilience, and other database-oriented features without extra programming effort and complexity.

The net result of combining SPPARKS with Stitch-IO is that a large part can be simulated using a small fraction (1/64th or smaller) of compute resources required to hold all of the data while running in the same wall clock time as if the whole simulation domain were deployed on a large cluster. Most importantly, computationally impossible domains that exceed the memory requirements of even the largest machines can now be run on a laptop with lossless results.

4. Stitch-CAD Overview. Stitch-CAD combines SPPARKS and Stitch-IO with Slic3r [5] to create digital twins. *Slic3r* is open source and easily accessible from the shell. Like a physical 3D printer, *Stitch-CAD* takes an STL file, defining the component geometry of a part, as input. *Slic3r* then slices the part into layers along the build axis (normally the z-axis), and defines infill lines for each layer in the form of G-code. The infill line direction is perpendicular to the infill lines of bordering layers. Hatch spacing and line angle can be modified in the *Slic3r* input file. (See Figure 4.1).

Since the focus here is on AM for metals (SLS printing), whereas *Slic3r* is designed to generate G-code for fused deposition modeling (FDM) or fused filament fabrication (FFF) (FDM/FFF) type printing, some processing of G-code is required. FDM/FFF printers require a continuous path, whereas metal SLS printers have the capability to turn the laser on and off. Before simulation, *Stitch-cad* removes these extraneous paths from the G-code, and extracts a list of the essential laser path movements for each layer. This is illustrated in Figure 4.2. *Stitch-CAD* then further breaks down the G-code paths into computational volumes that can be processed reasonably with a minimum of idle compute resources.

Stitch-Cad generates a separate computational volume for each path line in the layer, representing the region affected by the laser path; Micro-structures only evolve within the HAZ (heat affected zone). In the case that the computational volume around a particular laser path line requires more than the maximum number of sites available (user specified), the path line is recursively split up into smaller segments.

The HAZ cap, HAZ tail and HAZ width measurements are used to calculate the com-

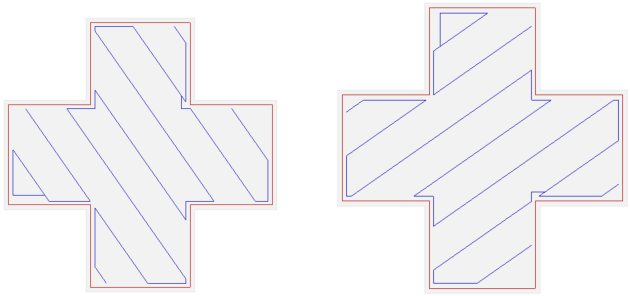


Fig. 4.1: Depiction of the Slic3r generated infill path lines (blue lines) for alternating layers of a plus cube with a fill angle of 35°. Infill pathlines are unpruned. Red outline represents perimeter.

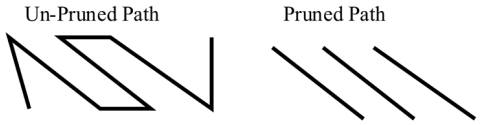
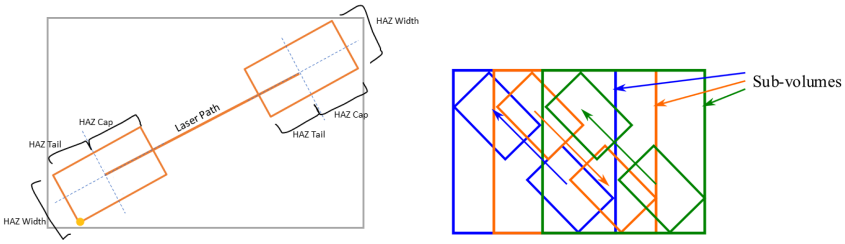


Fig. 4.2: Example of the Slic3r generated G-code path and then the pruned version appropriate for SLS printing.

putational volume. The laser path is shortened so that HAZ boundaries do not extend past endpoints of the original path line.



(a) Computational Volume (b) Sub-Volume Collection

Fig. 4.3: 4.3(a) demonstrates how computational volumes are calculated using HAZ Width, HAZ Tail, and HAZ Cap dimensions along with start point, endpoint and laser path vector. Computational volume is denoted by gray box. 4.3(b) depicts how computational volume moves with laser path.

Each laser path is contained within a computational volume; each computational volume is sized according the HAZ parameters. SPPARKS simulates micro-structure formation and evolution on the laser path within each computational volume, and results are incrementally appended to a *Stitch* file, similar to the way material would be added to an AM part. The computational volume and sub-volume collection are illustrated in Figure 4.3.

4.1. Input Parameters. The SPPARKS *potts/additive* app style was used to simulate micro-structures. A length scale parameter c, with units of μ (microns) per site, is used to relate physical length scale dimensions of microns to SPPARKS lattice site dimensions; for simulations results presented in this report, $c = 10$ was used. A *Slic3r Fill Density* of 20% was used so that there was around 50% overlap between adjacent computational volumes.

Table 4.1: Slic3r Specifications

Layers and Perimeters	
Layer Height	0.03 mm
First Layer Height	0.03mm
Infill	
Fill Density	20 %
Fill Pattern	Rectilinear
Fill Angle	0°
Perimeters	0

Table 4.2: SPPARKS Laser Specifications

SPPARKS Laser Specifications		
	Microns	Sites
spot width	100	10
melt tail length	100	10
melt depth	40	4
cap height	50	5
HAZ	130	13
tail HAZ	130	13
depth HAZ	60	6
cap HAZ	60	6
layer thickness	30	3

5. Algorithm. For each layer, *Stitch-Cad* extracts pruned laser paths. For each laser path, *Stitch-Cad* computes the computational volume, initializes the region, then performs the SPPARKS simulation for that defined laser path and appends simulation results to *Stitch* database. An outline of the *Stitch-Cad* algorithm is listed below.

1. Split G-code into layers
2. Prune each layer
3. For each laser path
   (a) Computational volume is calculated
   (b) Computational volume is initialized
   (c) SPPARKS simulation on laser path using *potts/additive*.

6. Main Results. Demonstration simulations were conducted on a staircase shaped domain; staircase dimensions are defined in Figure 6.1.

Stitch-Cad generated paths for the first two layers in staircase build simulation are shown in Figure 6.2; as shown, scan patterns alternate in direction, e.g. direction x on layer 1 followed by direction y on layer 2 and so on for entire build.

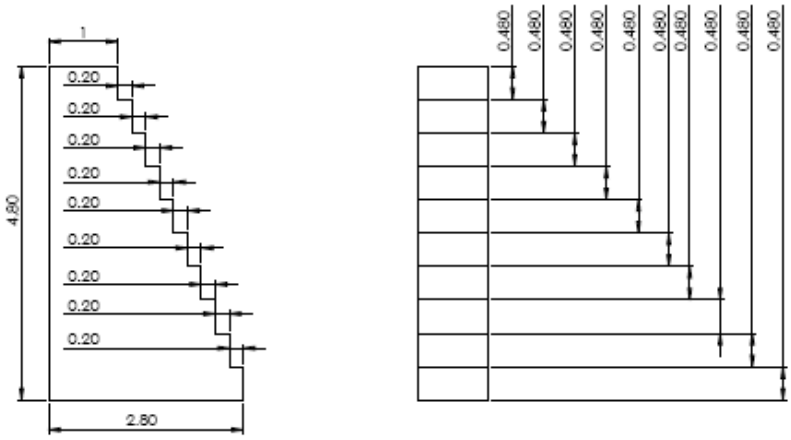


Fig. 6.1: Detailed drawing of staircase dimensions, in mm

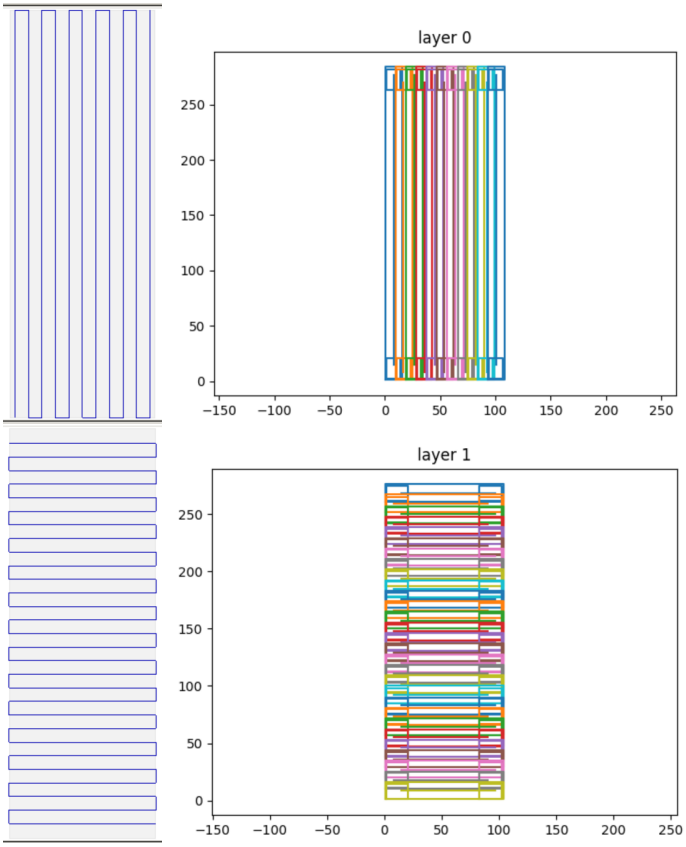
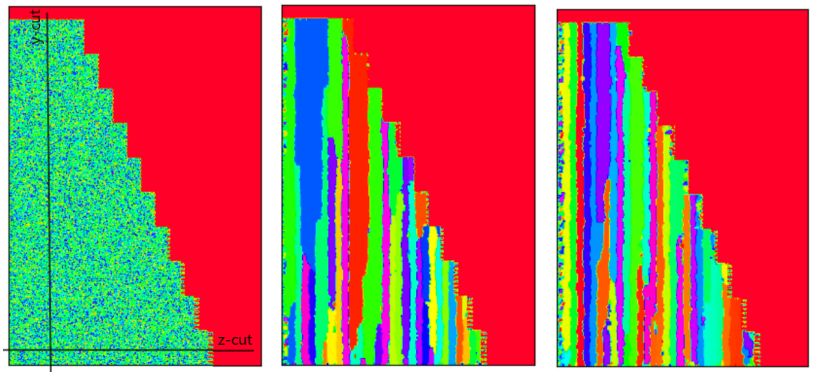


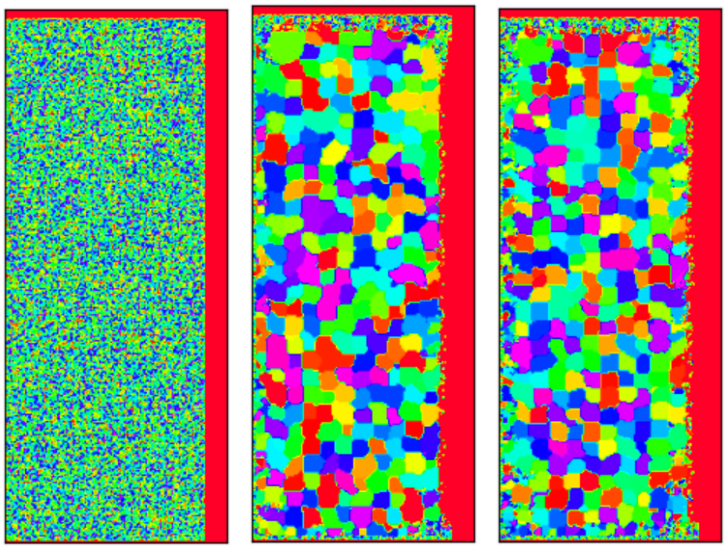
Fig. 6.2: *Slic3r* generated infill lines (left) and corresponding computational volumes and pruned path lines (right) generated by *Stitch-Cad*.

Simulated micro-structures on the staircase are shown in Figure 6.3; top image depicts randomized initialization of micro-structure on SPPARKS lattice and graphical annotation of y and x cuts; lower two images depict effect of a variable laser speed. For the process parameters used, columnar grains appear along build direction arising from epitaxial grain growth. $z-$cut and $y-$cut images of simulated build are shown in Figures 6.4 and 6.5.



(a) initialization (b) speed = 9 sites/mcs (c) speed = 12 sites/mcs

Fig. 6.3: x-cut ($x = 50$). Lower two images depict predicted micro-structures for two different laser speeds. Comb structure at right edges are numerical and graphical artifacts due to rendering and handling of geometry.



(a) initialization (b) speed = 9 sites/mcs (c) speed = 12 sites/mcs

Fig. 6.4: z-cut ($z = 25$); two different laser speed. Right edge depicts numerical and graphical rendering artifacts related to handling of paths and geometry.

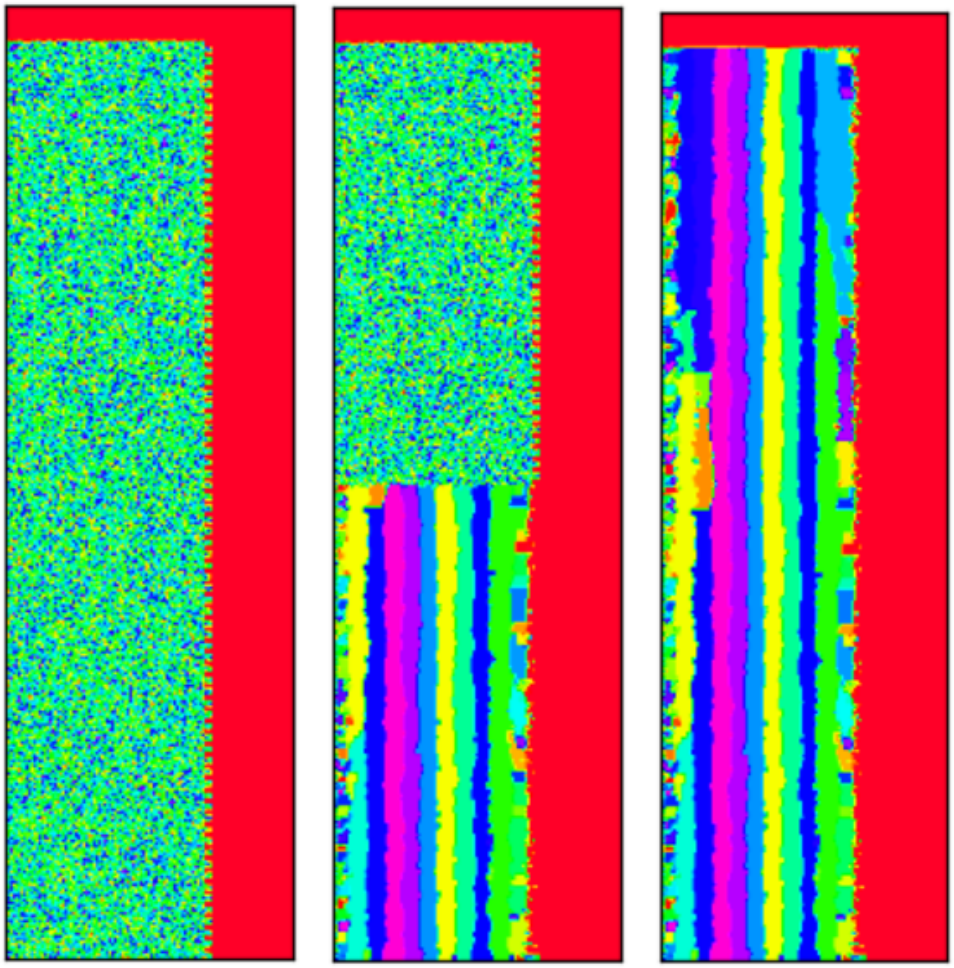


Fig. 6.5: y-cut ($y = 50$). Depicts build, for speed $12 sites/mcs$, halfway through the process (middle) and at end of process (right). Right edge depicts numerical and graphical rendering artifacts related to handling of paths and geometry.

Figure 6.6 demonstrates how the *Stitch-Cad* output compares to a SPPARKS simulation staircase with the same dimensions (see Table 4.2) built without *Stitch-Cad*. In this second build, each layer was simulated in its entirety, using a serpentine laser pattern with a hatch spacing of 50 microns (5 sites) on each layer, and a speed of 14 sites/mcs. These path lines were very similar to the *Slic3r* generated infill lines used in our *Stitch-Cad* build. Both have largely vertical grain growth but the *Stitch-Cad* build has narrower crystals.

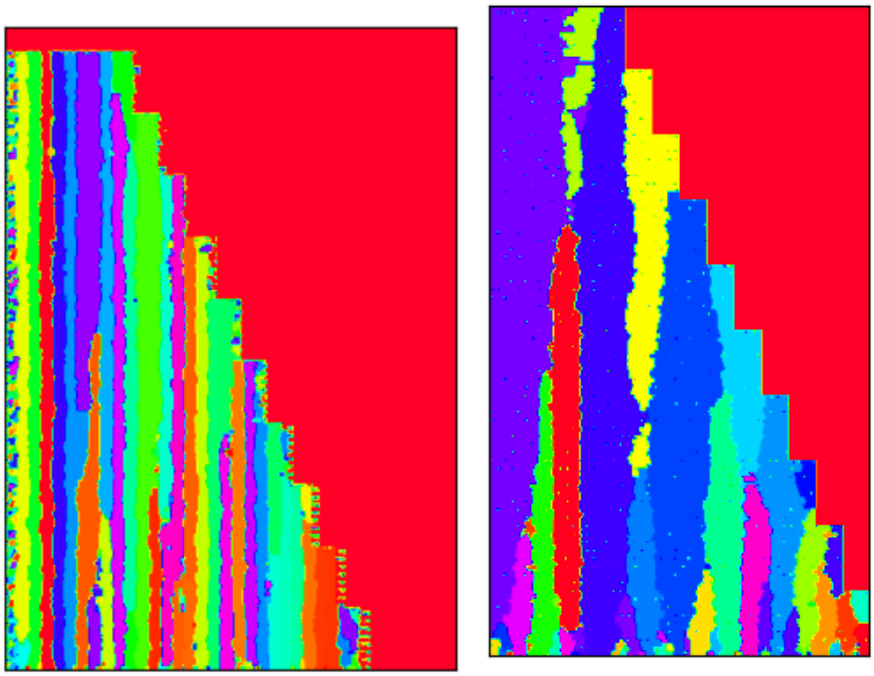
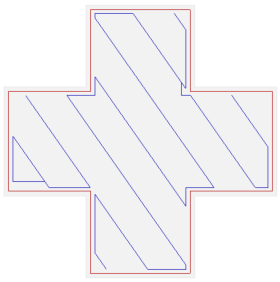


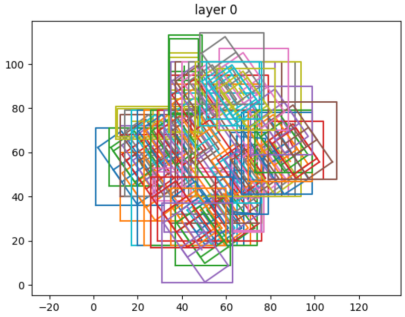
Fig. 6.6: x cut stitch-Cad and example SPPARKS.

6.1. Geometry and initialization. SPPARKS simulations are conducted on a sequence of unique computational volumes to assemble a complex geometry. Each computational volume is axis aligned as defined by laser path lines which are not necessarily axis aligned or conformal to the exact STL defined geometry. The consequence of this is shown in Figure 6.7. Note geometry implied by upper right and bottom images in Figure 6.7 – this geometry does not reflect the actual geometry shown in upper left image. Computational volumes are not necessarily confined to the interior of the STL defined volume; however laser path lines remain inside the STL defined geometry. As another example, consider overhang artifacts in lower steps of staircase build simulation shown in Figures 6.5 and 6.8. This is due to alternating direction of path lines between adjacent layers. Infill lines are layer dependent.

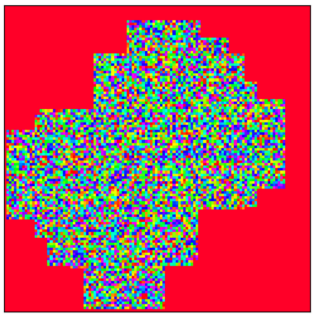
In some cases, there is a larger distance between the *Slic3r* generated infill lines and the perimeter which can cause a larger variation in overall layer dimensions after computational volumes are combined on the layer. This issue may be resolved by considering how computational volumes are computed from *Slic3r* GCODE paths; clipping of domain prior to rendering may also be an important tool to eliminate these artifacts. Because of these practical facts, rendering issues are created which are not necessarily simulation problems but rather rendering artifacts; future work is planned to develop methods for clipping simulated results for the purposes of rendering and further analysis. This could also potentially be resolved using the perimeter coordinates, which are also specified in the G-code.



(a) Geometry and *Slic3r* z-cut path lines



(b) *Stitch-Cad* generated sequence of computational volumes on layer; image also includes bounding box renderings



(c) Rendering of volume initialized for SPPARKS simulation on layer

Fig. 6.7: Plus-cube. Geometry, computational volumes and initialization.

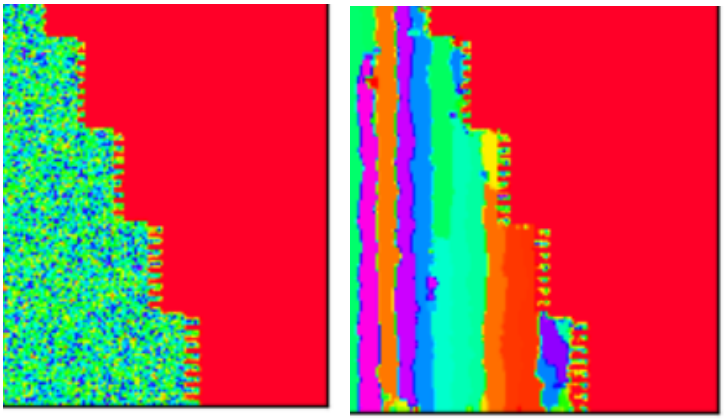


Fig. 6.8: Before and after of overhangs, on lower steps of staircase.

Another issue to note is that path line coordinates in $x-y$ plane on a layer are normalized by *Stitch-Cad* using minimum and maximum coordinate values on the layer. For *Stitch-Cad* to properly generated computational volumes, all layers must have the same $x-y$ origin

otherwise the layer geometry defined by STL would be incorrectly modified by *Stitch-Cad*. This condition was already satisfied by the staircase STL file. However, for more complex geometries, this is something that must be addressed otherwise path lines generated by *Stitch-Cad* will not strictly adhere to geometry defined in STL file.

7. Conclusions. Using *Stitch-Cad* we have demonstrated the ability to simulate micro-structures on more complex geometries defined by STL files. *Slic3r* was used to slice the STL defined geometry into layers and generate raster scan patterns defined by GCODE. Using *Stitch-Cad*, GCODE was interpreted into suitable SPPARKS commands for simulating micro-structure formation and evolution on the STL defined geometry. This work represents the first demonstration simulating an AM build in SPPARKS using non-trivial geometries defined by an STL file. This work also identified an important computational issue which much be addressed to more generally handle STL geometries; the local coordinate system origin on each layer must be carefully considered to handle general complex geometries; this will be the subject of future and continuing work. Rendering of micro-structures is an essential element of this work – another important finding is that micro-structure renderings must respect STL defined geometry otherwise usefulness of visualization and process rendering is diminished. As this project progresses, this issues will be addressed as well exploring more complex models and the issues they may reveal.

REFERENCES

[1] J. Lofstead, J. Mitchell, and E. Chen, *Stitch it up: Using progressive data storage to scale science*, in In Proceedings of IPDPS'20, May 25-29, 2020.
[2] S. Plimpton, A. Thompson, and A. Slepoy, *SPPARKS*. http://spparks.sandia.gov/index.html, 2009.
[3] S. Plimpton, A. Thompson, and A. Slepoy, *SPPARKS*, 2016. http://spparks.sandia.gov.
[4] S. Plimpton et al, *Crossing the mesoscale no-man's land via parallel kinetic Monte Carlo*, Tech. Report SAND2009-6226, Sandia National Laboratories, 2009.
[5] A. Ranellucci, *Reprap/slic3r and the future of 3d printing*, Canessa, E., Fonda, C., and Zennaro, ed.,"Low-cost 3D Printing for Science, Education, and Sustainable Development, (2013), pp. 75–82.
[6] T. M. Rodgers, J. A. Mitchell, and V. Tikare, *A Monte Carlo model for 3D grain evolution during welding*, Modelling and Simulation in Materials Science and Engineering, 25 (2017), p. 064006.

LEARNING COMPACT PHYSICS-AWARE DELAYED PHOTOCURRENT MODELS USING DYNAMIC MODE DECOMPOSITION

JOSHUA HANSON*, PAVEL BOCHEV†, AND BILIANA PASKALEVA‡

Abstract. Radiation-induced photocurrent in semiconductor devices can be simulated using complex physics-based models, which are accurate, but computationally expensive. This presents a challenge for implementing device characteristics in high-level circuit simulations where it is computationally infeasible to evaluate detailed models for multiple individual circuit elements. In this work we demonstrate a procedure for learning compact delayed photocurrent models that are efficient enough to implement in large-scale circuit simulations, but remain faithful to the underlying physics. Our approach utilizes Dynamic Mode Decomposition (DMD), a system identification technique for learning reduced-order discrete-time dynamical systems from time series data based on singular value decomposition. To obtain physics-aware device models, we simulate the excess carrier density induced by radiation pulses by solving numerically the Ambipolar Diffusion Equation, then use the simulated internal state as training data for the DMD algorithm. Our results show that the significantly reduced-order delayed photocurrent models obtained via this method accurately approximate the dynamics of the internal excess carrier density—which can be used to calculate the induced current at the device boundaries—while remaining compact enough to incorporate into larger circuit simulations.

Key words. Dynamic Mode Decomposition, Ambipolar Diffusion Equation, delayed photocurrent, machine learning, data-driven compact models.

1. Introduction. Ionizing radiation can affect operation of electronics in multiple application contexts including space, terrestrial and manmade nuclear environments. Specifically, fluence of ionizing radiation generates electron-hole pairs within semiconductor devices, resulting in excess currents flowing through the devices. Such excess current is not present during device operation in a normal environment and its impact on electronic systems can be catastrophic, ranging from instantaneous interruptions in service, loss of stored memory, and even burnout of the entire system. For example spacecraft depend on electronic components that must perform reliably over missions measured in years and decades and space radiation is a primary source of degradation, reliability issues, and potentially failure for these electronic components. Physics-based modeling and simulation of radiation effects on electronic systems in various radiation environments can facilitate understanding of the mechanisms governing the radiation response of the electronic materials, parts, and systems, and can be used to devise ways to mitigate radiation effects, and create new materials and devices that are resilient to radiation exposure.

Thus, computational analysis of radiation effects on electronic systems has utility ranging from guiding the initial designs of systems, setting up the design of experiments, and final qualification. At a device level the excess carrier behavior can be accurately modeled by the Drift-Diffusion equations (DDE) [17] given by

$$\nabla \cdot (\epsilon \mathbf{E}) = q(p - n + C) \tag{1.1}$$

$$\frac{\partial n}{\partial t} = \nabla \cdot (n\mu_n \mathbf{E} + D_n \nabla n) - R + g \tag{1.2}$$

$$\frac{\partial p}{\partial t} = \nabla \cdot (p\mu_p \mathbf{E} + D_p \nabla p) - R + g \tag{1.3}$$

where n and p are the concentrations of the electrons and holes, respectively, μ_n, μ_p, D_p, and D_n are the carrier mobilities and diffusivities, C is the doping concentration, $\mathbf{E}$ is the

*University of Illinois at Urbana-Champaign, jmh4@illinois.edu
†Sandia National Laboratories, pbboche@sandia.gov
‡Sandia National Laboratories, bspaska@sandia.gov

electric field, q is the electron charge, R is the recombination rate, and g is the carrier generation density.

Numerical solution of (1.1)–(1.3) using, e.g., a multi-dimensional finite element discretization of the device, forms the basis of the so-called Technology Computer-Aided Design (TCAD) simulators. However, (1.1)–(1.3) is a coupled system of nonlinear Partial Differential Equations (PDEs) whose accurate numerical solution can be very time consuming. As a result, although TCAD device simulators based on the DDE could in principle be coupled to circuit simulators, their high computational cost makes the analysis of all but very small circuits computationally intractable.

At the other end of the spectrum are the so-called compact device models, which are computationally efficient but rely on empirical approximations and/or simplified analytic solutions to the semiconductor transport equations. Often such models must be recalibrated for different operating regimes and inclusion of new physics may force redevelopment of the model from scratch.

Data-driven techniques present an opportunity to automate the development of compact models by learning them directly from laboratory measurements and/or suitable synthetic data. For such models to be practically useful though, they must be able to correctly predict the device response when the device is integrated into a circuit and exposed to a wide range of stimuli. The caveat is that in a laboratory setting one can only apply a limited type of signals and directly measure the device response, leading to "sparse" training sets. In contrast, traditional Machine Learning (ML) applications, such as natural language processing and image classification [11, 10, 8] operate in "big data" environments. As a result, a compact device model learned solely from such "sparse" data may fail to generalize to all relevant analysis conditions because it will learn salient patterns in the dataset rather than the causal physics underpinning the device operation.

The latter, i.e., the fact that device behavior is governed by strict physics laws can be used to counter the lack of big data by incorporating the physics knowledge into the model development. Recent work on scientific Machine Learning [14, 13, 2] suggests this strategy is effective and can lead to generalizable models.

The main goal of this work is to establish the viability of physics-aware machine learning for the development of compact data-driven photocurrent models that are efficient enough to allow large circuit simulations, while remaining faithful to the basic physics principles embodied by models such as (1.1)–(1.3). To that end we adopt a setting that has been used extensively in the past five decades at Sandia National Laboratories for the development of compact analytic photocurrent models. Although this setting uses a simplified version of the DDE, it provides a stepping stone towards future developments of data-driven models based on the fully coupled system (1.1)–(1.3). In so doing we are able to leverage a wealth of experiences and physics knowledge accumulated through the use of the existing compact models, as well as provide a reference point for evaluation of our data-driven models.

We have organized the paper as follows. Section 2 reviews the current state-of-the art in compact photocurrent models and establishes the physical basis for our data-driven model. Section 3 briefly summarizes the finite element discretization of the physics model used to generate synthetic training data. The core of the paper is Section 4 where we use Dynamic Mode Decomposition ideas [16, 12, 7] to develop our model. Section 5 contains computational results highlighting the performance of the model as well as comparison with published results in [1]. We summarize our findings and discuss future research in Section 6.

2. Physics-Based Compact Analytic Photocurrent Models. Although the DDE (1.1)-(1.3) accurately describes the behavior of the excess carriers, it does not lend itself to

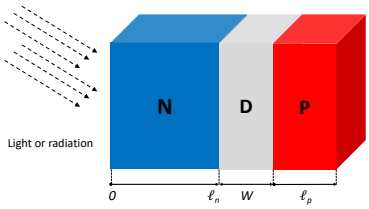


Fig. 2.1: Traditional compact photocurrent model development splits each PN-junction within a device into a depletion region D having width W, and quasi-neutral P and N regions with lengths ℓ_n and ℓ_p, respectively; see, e.g., [1].

exact analytical solution. To enable analytic approximation of the governing equations, most photocurrent models in use today follow the same basic approach as in the classic paper [19] and split each PN-junction within a device into a depletion region and quasi-neutral P- and N-regions; see Fig. 2.1. Carriers in the depletion region are quickly converted to photocurrent and yield the so-called prompt photocurrent I_{prompt}. Carriers in the P and N regions have a delayed response and produce the *delayed* photocurrents I_p and I_n, respectively. As a result, the total photocurrent is given by

$$I_{\text{total}} = I_{\text{prompt}} + I_p + I_n \,.$$

To calculate these currents one makes additional simplifying assumptions. The first is the electrical neutrality approximation, which stipulates that the excess electron and hole densities are equal throughout the entire device. The second is the congruence assumption, which states that the electron and hole fluxes into or out of any region must be equal; see, e.g., [9]. Under these assumptions the DDE model (1.1)-(1.3) in the P and N regions can be replaced by the Ambipolar Diffusion Equation (ADE)

$$\frac{\partial u}{\partial t} = D_a \nabla^2 u - \mu_a \mathbf{E} \cdot \nabla u - \frac{1}{\tau_a} u + g \tag{2.1}$$

where u, D_a, μ_a and τ_a are the excess carrier density (electrons or holes), the ambipolar diffusion coefficient, the ambipolar mobility and the carrier lifetime, respectively. In general, these parameters may depend on the excess carrier density and, as a result, (2.1) is still a nonlinear PDE. However, for moderate radiation dose rates these coefficients can be approximated by constant values and the ADE becomes a linear parabolic PDE. The final assumption is that the depletion region width W is not affected by the excess carriers and is a constant. Under these assumptions, the prompt photocurrent is modeled as

$$I_{\text{prompt}} = qgAW \,,$$

where A is the effective area of the PN-junction, while I_p and I_n are modeled by the ADE (2.1).

At this point traditional compact photocurrent model development proceeds with deriving analytic approximations for the solutions of (2.1) in one dimension and using them to obtain expressions for the delayed photocurrents. Early work [19] considered unbounded P and N regions and negligible electric fields. The resulting Wirth-Rogers model tends to overestimate the photocurrent as it neglects the effects of an ohmic contact at a finite distance from the depletion region [1]. Subsequent work [5] relaxed these conditions, assumed that $\mathbf{E}$ is constant, and used approximate Laplace transforms to obtain analytic expressions for I_p and I_n. However, approximation of the Laplace transform results in a model that

yields unphysical current estimates when $\mathbf{E}$ exceeds roughly $10\,\mathrm{V\,cm^{-1}}$; see [20]. The latter work used Fourier analysis techniques to develop a more accurate photocurrent model that avoids these drawbacks. The model in [20] was further improved in [1] by using a transformation of (2.1) into an inhomogeneous heat equation and solving the latter exactly by Fourier techniques. Another popular photocurrent model is the Fjeldly Model [6]. Unlike the Axness-Kerr model [1], which solves the time-dependent equation (2.1) exactly, the Fjeldly Model uses a steady-state solution of ADE combined with an RC delay circuit to achieve time dependence.

In this work we adopt the above setting and focus on the development of physics-aware compact data-driven models for the delayed photocurrents in the P- and N-regions that can be used as "plug-and-play" substitutes for conventional models. The core idea is to replace the *simplified analytic solutions* of the ADE, comprising the basis of most standard compact models, by a *discrete-time dynamical system* approximating the flow map of the ADE, i.e. the "solution operator" that returns the internal device state at a given time as a function of the history of states and external input values occuring at all previous times. In other words, instead of approximating the analytic solution directly, the end model produces an approximate solution via a system of difference equations.

To that end we apply a Dynamic Mode Decomposition (DMD) [16] approach to samples of the internal state (carrier density) of the device obtained by solving (2.1) numerically by a Finite Element Method (FEM). In this paper we assume that all device parameters such as diffusion coefficients, carrier lifetimes and doping concentrations are known. In this case ADE already contains all the necessary physics information and development of physics-aware compact data driven models can be done entirely from synthetic data[1].

A more general setting occurs when one or more of the ADE parameters are either unknown or have large uncertainties. In this case the development of the compact model must also include a parameter identification step to refine the ADE, which requires laboratory data. Since the purpose of this paper is to demonstrate the viability of DMD as an effective tool for the generation of compact device models, detailed discussion of this more general setting is beyond the scope of this paper and will be addressed in a forthcoming work.

Regardless of the particular setting though the physics-awareness of our models stems from the fact that they represent *approximations of the flow map engendered by the physics model*, and built from *simulated internal states* of this model. These states contain physics information that cannot be obtained by laboratory instruments, which can typically only measure the currents at the device terminals. The latter may not be "rich enough" for a traditional ML approach, as well as for DMD, to obtain a reliable model of the underlying causal physics.

3. Numerical solution of the ADE. We consider the ADE on the space-time domain $\Omega := \mathcal{X} \times \mathcal{T} \subset \mathbb{R}^2$, where $\mathcal{X} := (0, \ell)$ and $\mathcal{T} := (0, t_{\text{final}})$. Without loss of generality, one may assume that $\mathcal{X}$ is the N-region of the device; see Fig. 2.1 and so we set $\ell = \ell_n$. To obtain a well-posed problem, we augment equation (2.1) with homogeneous initial and boundary conditions, i.e.,

$$u(x,0) = 0 \quad \forall x \in \mathcal{X} \quad \text{and} \quad u(0,t) = u(\ell,t) = 0 \quad \forall t \in \mathcal{T}\,.$$

The homogenous initial condition corresponds to the fact that at $t = 0$ there are no excess carriers present in the device. The boundary condition choice corresponds to assuming infinite carrier recombination velocities and ohmic contacts at $x = 0$ and the boundary of

[1] In this setting the resulting compact model can be interpreted as a discrete reduced-order model of the flow map induced by the ADE.

the depletion region $x = \ell$; see [1]. More general boundary conditions not requiring these assumptions can also be considered but are not necessary for the purpose of this work.

We will simulate the internal state of the device by using the method of lines to solve the ADE numerically. For the spatial discretization we consider a standard Galerkin finite element method and then solve the resulting system of Ordinary Differential Equations (ODEs) using an implicit numerical integration scheme. For completeness we briefly review the discretization process below.

Let $\mathcal{X}^h$ denote a uniform[2] partition of $\mathcal{X}$ into $n+1$ elements κ_i with vertices $\{x_i\}_{i=0}^{n+1}$, i.e., $\kappa_i = [x_i, x_{i+1}]$, $i = 0, \ldots, n$, $x_0 = 0$ and $x_{n+1} = \ell$. The mesh parameter is given by $h = \frac{\ell}{n+1}$.

As usual, $L^2(\mathcal{X})$ denotes the space of all square integrable functions on $\mathcal{X}$ with norm and inner product denoted by $\|\cdot\|_0$ and $(\cdot,\cdot)_0$, respectively, and $H_0^1(\mathcal{X})$ is the Sobolev space of order one whose elements are constrained to vanish at the boundary points. The weak variational form of the ADE is then given by *seek* $u \in H_0^1(\mathcal{X})$ *such that*

$$(u_t, v)_0 + Q(u, v) = (g, v)_0 \quad \forall v \in H_0^1(\mathcal{X})\,. \tag{3.1}$$

The bilinear form $Q(\cdot,\cdot) : H_0^1(\mathcal{X}) \times H_0^1(\mathcal{X}) \mapsto \mathbb{R}$ is defined as

$$Q(u, v) = D_a\,(u_x, v_x)_0 + \mu_a \mathbf{E}\,(u_x, v)_0 + \frac{1}{\tau_a}\,(u, v)_0\ . \tag{3.2}$$

To discretize (3.1) in space we consider a nodal (Lagrangian) conforming finite element subspace $V_0^h \subset H_0^1(\mathcal{X})$; see, e.g., [4]. Let $\{v_i\}_{i=1,n}$ be the standard nodal basis having the property that $v_i(x_j) = \delta_{ij}$. We then seek an approximate solution of (2.1) as

$$u_h(x, t) = \sum_{i=1}^{n} u_i(t) v_i(x)\,, \tag{3.3}$$

where $\mathbf{u}(t) = (u_1(t), \ldots, u_n(t)) \in \mathbb{R}^n$ is a vector of unknown solution coefficients. Inserting (3.3) into the weak form (3.1) and restricting the test space to V_0^h then yields the system of ODEs

$$M\dot{\mathbf{u}}(t) + K\mathbf{u}(t) = \mathbf{g} \tag{3.4}$$

where $M, K \in \mathbb{R}^{n\times n}$ are the (consistent) finite element mass and stiffness matrices with elements

$$M_{ij} = (v_i, v_j)_0 \quad \text{and} \quad K_{ij} = Q(v_i, v_j)\,,$$

respectively, and $\mathbf{g}(t) \in \mathbb{R}^n$ is a discrete source term with $g_i(t) = (g, v_i)_0$.

The ODE system (3.4) can be solved by any standard time-integration scheme. However, in general (3.4) is stiff and an implicit scheme is preferred. In this paper, we use an implicit multi-step variable-order routine based on a backward differentitation formula (BDF) for approximating the state derivative; for more details see [18]. This method is included by default in the `scipy.integrate` submodule within the SciPy v1.5.1 package for Python 3.

[2]Utilizing a variable mesh spacing with increased node density near the boundary may provide some potential advantages such as more accurate gradient estimation at the edge points, however in the context of this work we restrict our attention to uniform meshes.

4. A Dynamic Mode Decomposition Photocurrent Model. Assume that an approximate solution of the ODE (3.4) derived via FEM discretization is available at uniformly spaced time steps $t_k := k\Delta t$, $k = 0, \ldots, m$ for some sampling interval $\Delta t > 0$. Then, the approximate numerical solution of the ADE can be represented as a linear discrete-time dynamical system describing the evolution of samples of the state $\mathbf{u}$ due to exogeneous input $\mathbf{g}$, i.e.,

$$\mathbf{u}_{k+1} = A\mathbf{u}_k + B\mathbf{g}_k \tag{4.1}$$

where

$$\mathbf{u}_k := [u_1(k\Delta t) \cdots u_n(k\Delta t)]^\mathsf{T} \tag{4.2}$$

$$\mathbf{g}_k := [g_1(k\Delta t) \cdots g_n(k\Delta t)]^\mathsf{T} \tag{4.3}$$

and $A, B \in \mathbb{R}^{n\times n}$ are linear maps. Expressing the system in this form facilitates the use of many familiar system identification techniques. In particular, Dynamic Mode Decomposition (DMD) is a data-driven method for learning the maps A and B from a time series of state and input measurements $\{\mathbf{u}_k, \mathbf{g}_k\}_{k=0}^m$, with the goal of identifying a small number of dominant dynamic modes.

Below we summarize the DMD algorithm adapted for control inputs as described in [12]. By organizing the samples into the following matrices

$$X = \begin{bmatrix} | & & | \\ \mathbf{u}_0 & \cdots & \mathbf{u}_{m-1} \\ | & & | \end{bmatrix}; \quad X' = \begin{bmatrix} | & & | \\ \mathbf{u}_1 & \cdots & \mathbf{u}_m \\ | & & | \end{bmatrix}; \quad \text{and} \quad G = \begin{bmatrix} | & & | \\ \mathbf{g}_0 & \cdots & \mathbf{g}_{m-1} \\ | & & | \end{bmatrix}, \tag{4.4}$$

we can express the linear relationships within the data as

$$X' = AX + BG = \begin{bmatrix} A & B \end{bmatrix} \begin{bmatrix} X \\ G \end{bmatrix} =: \begin{bmatrix} A & B \end{bmatrix} S. \tag{4.5}$$

Therefore the maps A and B can be approximated by

$$\begin{bmatrix} A & B \end{bmatrix} \approx \begin{bmatrix} \bar{A} & \bar{B} \end{bmatrix} := X'S^\dagger \tag{4.6}$$

where $\dagger$ indicates the Moore-Penrose pseudoinverse. An efficient and accurate algorithm for estimating the pseudoinverse of a rectangular matrix is realized via truncated singular value decomposition. The matrix of samples S can be factored as

$$S = U\Sigma V^\mathsf{T} = \begin{bmatrix} \tilde{U} & \tilde{U}_{\text{trun}} \end{bmatrix} \begin{bmatrix} \tilde{\Sigma} & 0 \\ 0 & \tilde{\Sigma}_{\text{trun}} \end{bmatrix} \begin{bmatrix} \tilde{V}^\mathsf{T} \\ \tilde{V}_{\text{trun}}^\mathsf{T} \end{bmatrix} \approx \tilde{U}\tilde{\Sigma}\tilde{V}^\mathsf{T} \tag{4.7}$$

where $U \in \mathbb{R}^{n\times n}$, $\tilde{U} \in \mathbb{R}^{n\times p}$, $\Sigma \in \mathbb{R}^{n\times m}$, $\tilde{\Sigma} \in \mathbb{R}^{p\times p}$, $V^\mathsf{T} \in \mathbb{R}^{m\times m}$, and $\tilde{V}^\mathsf{T} \in \mathbb{R}^{p\times m}$. Here $_{\text{trun}}$ denotes the $m - p$ truncated singular values, so that the p greatest singular values are kept. In practice, one sets an error tolerance and truncates all singular values below this threshold. Now the pseudoinverse of S is naturally approximated by

$$S^\dagger = \tilde{V}\tilde{\Sigma}^{-1}\tilde{U}^\mathsf{T} \tag{4.8}$$

Substituting equation (4.8) into (4.6) gives

$$A \approx \bar{A} \approx X'\tilde{V}\tilde{\Sigma}^{-1}\tilde{U}_1{}^\mathsf{T} \in \mathbb{R}^{n\times n} \tag{4.9}$$

$$B \approx \bar{B} \approx X'\tilde{V}\tilde{\Sigma}^{-1}\tilde{U}_2{}^\mathsf{T} \in \mathbb{R}^{n\times n}, \tag{4.10}$$

where $\tilde{U} = \begin{bmatrix} \tilde{U}_1^\mathsf{T} & \tilde{U}_2^\mathsf{T} \end{bmatrix}^\mathsf{T}$ with $\tilde{U}_1 \in \mathbb{R}^{n\times p}$, $\tilde{U}_2 \in \mathbb{R}^{n\times p}$.

4.1. Projection-Based Model Order Reduction. We can achieve a more compact model by incorporating the projection $\tilde{\mathbf{u}} = P\mathbf{u}$ of the state onto the canonical dynamic mode coordinates. In the same manner as equation (4.7), we factor the matrix $X' \approx \hat{U}\hat{\Sigma}\hat{V}^{\mathsf{T}}$, where the truncations are chosen to preserve the r greatest singular values. Then the projection onto dynamic mode coordinates is given simply by $P = \hat{U}^{\mathsf{T}} \in \mathbb{R}^{r\times n}$. The final reduced-order model obtained via DMD is the discrete-time dynamical system:

$$\tilde{\mathbf{u}}_{k+1} = \tilde{A}\tilde{\mathbf{u}}_k + \tilde{B}\mathbf{g}_k \tag{4.11}$$

$$\tilde{\mathbf{u}}_0 = \hat{U}^{\mathsf{T}}\mathbf{u}_0 \tag{4.12}$$

$$\mathbf{u}_k \approx \hat{U}\tilde{\mathbf{u}}_k, \tag{4.13}$$

where

$$\tilde{A} := \hat{U}^{\mathsf{T}}\bar{A}\hat{U} = \hat{U}^{\mathsf{T}}X'\tilde{V}\tilde{\Sigma}^{-1}\tilde{U_1}^{\mathsf{T}}\hat{U} \in \mathbb{R}^{r\times r} \tag{4.14}$$

$$\tilde{B} := \hat{U}^{\mathsf{T}}\bar{B} \quad = \hat{U}^{\mathsf{T}}X'\tilde{V}\tilde{\Sigma}^{-1}\tilde{U_2}^{\mathsf{T}} \quad \in \mathbb{R}^{r\times n}. \tag{4.15}$$

The parameter p represents the number of dynamic modes to fit to the data, which controls the model precision. If a quantitative uncertainty analysis is desired, the Eckart-Young theorem provides an avenue to bound the truncation error at this step as a function of p. The parameter r represents the number of modes to project onto, that is, the order of the final reduced-order model, which controls the model compactness. The case where $r > p$ usually results in diminished performance; $r = p$ retains exactly the same number of modes fit to the data in the compactified model; $r < p$ results in a more compact model, but ignores the $p - r$ least significant modes fit to the data, which can result in slightly reduced accuracy. In this work we will use $r = p$.

REMARK 1. *Our DMD model is constructed from synthetic data obtained by a numerical solution of the ADE. Although traditional Reduced Order Models (ROM) also use synthetic data generated by a "full order model" (FOM), there are some important distinctions between such models and the approach in this paper. Specifically, ROM uses the synthetic data (the "snapshots") to define a reduced order trial basis and then finds the reduced order solution by either projecting the FOM onto that basis or performing a least-squares residual minimization [3, 15]. For transient problems state snapshots have to be generated at different times to allow advancing the solution in time. In contrast, our approach directly learns a discrete solution operator that advances the solution without invoking the full order model. In particular, the DMD model can be equally well defined in the absence of a trusted FOM from bona fide laboratory measurements. In this paper we chose to use synthetic data because ADE is considered to be adequate for typical photocurrent applications of interest.*

4.2. Training of the compact model. Construction of the DMD model (4.11) requires training samples representing time series of state and input measurements $\{\mathbf{u}_k, \mathbf{g}_k\}_{k=0}^{m}$. To generate such samples we recall the assumption in Section 2 that all device parameters are known. In particular, here we consider a generic PN-junction device characterized by the parameters in Table 4.1. The values in this table are adapted from [1] to enable a direct comparison with a published compact photocurrent model.

We then select a suitable set of generation density functions $\{g^k_{\text{train}}\}_{k=1}^{M}$ and use the computational scheme in Section 3 to solve (2.1) numerically with homogeneous initial and Dirichlet boundary conditions. Selection of the inputs g^k_{train} depends on the type of the anticipated testing input(s) g_{test} for the model and will be revisited in Section 5.

Obtaining a numerical solution for the ADE requires proper scaling and non-dimensionalization of the governing equations. For convenience we scale the computational domain $\mathcal{X}$ so that

Table 4.1: ADE parameters for a generic PN-junction device

Parameter	Value	Units	Description
ℓ	3.075×10^{-2}	cm	N-region length
D_a	1.19×10^{1}	$\mathrm{cm}^2\,\mathrm{s}^{-1}$	diffusion coefficient
μ_a	4.64×10^{2}	$\mathrm{cm}^2\,\mathrm{V}^{-1}\,\mathrm{s}^{-1}$	typical hole mobility for Si
ℓ_n	1.54×10^{-2}	$\mathrm{cm}^2\,\mathrm{s}^{-1}$	diffusion length: $\sqrt{D_a \tau_a}$
τ_a	1.97×10^{-5}	s	typical hole lifetime
$\mathbf{E}$	-20 or 0	$\mathrm{V\,cm}^{-1}$	electric field
C	1×10^{17}	cm^{-3}	doping concentration
$\widehat{g}$	4.3×10^{22}	$\mathrm{cm}^{-3}\,\mathrm{s}^{-1}$	maximum generation density
$u(x,t)$	variable	cm^{-3}	excess carrier concentration (ADE solution)

the length of the N-region, i.e., ℓ, becomes a unit of length and t_{final} becomes a unit of time. After rescaling the domain and the equations the computational domain $\mathcal{X}$ becomes the unit square, and the non-dimensional ADE coefficients are given by

$$\begin{array}{ll} D_a = 0.063 \\ \mu_a = 0.063 \end{array} \; ; \quad \begin{array}{ll} L_p = 0.5 \\ \tau_a = 3.92 \end{array} \; ; \quad \text{and} \quad \begin{array}{ll} \mathbf{E} = -23.84 \text{ or } 0 \\ \widehat{g} = 2.15 \end{array} \; . \tag{4.16}$$

We highlight that the numerical solution of the ADE constitutes the "physics-based" element of our procedure. Specifically, the physics information is incorporated by using trusted *a priori* dynamics models—which are calibrated or driven by experimental measurements—to generate training data from simulating the unobservable internal state of a device using robust numerical techniques.

Remark 2. *In this work we use the ADE as the physical basis for the data-driven model because it has been used to develop almost all compact photocurrent models in use today, i.e., it is an example of model that is trusted based on decades-long practical experiences. However, we emphasize that the DMD algorithm is also suitable for more complex physics-based models, such as the full drift-diffusion equations or detailed molecular dynamics simulations, with the capability of producing dramatically reduced-order approximations that are feasible to implement in high-level circuit simulators but remain faithful to the underlying physics.*

5. Simulation Results. In the following section, we evaluate the performance of a DMD model (4.11) that has been trained according to the procedures described in Section 4. Our numerical testing process is as follows. Let $g(x,t)$ be a target generation density for which we seek the response of our device. We sample $g(x,t)$ in space using the vertices defining the finite element mesh $\mathcal{X}^h$, and in time using a desired time step Δt for a total of m time steps. This sampling produces the inputs $\mathbf{g}_k$ to the DMD model. We then set $\mathbf{u}_0 = \mathbf{0}$ and use (4.11) to recover the internal state of the device, i.e., the excess carrier concentration, at the mesh nodes $\{x_i\}_{i=1}^n$ for every $t_k = k\Delta t$:

$$\tilde{\mathbf{u}}_{k+1} = \tilde{A}\tilde{\mathbf{u}}_k + \tilde{B}\mathbf{g}_k\,, \quad k = 0, \ldots, m-1\,.$$

Each vector $\tilde{\mathbf{u}}_k$ induces a C^0 finite element function

$$u_h^{\mathrm{DMD}}(x,t_k) = \sum_{i=1}^{n} \tilde{u}_i(t_k) v_i(x) \tag{5.1}$$

which is the predicted internal carrier density. Using u_h^{DMD} and taking into account the homogeneous Dirichlet boundary condition $u_h^{\mathrm{DMD}}(x_0,t_k) = u_h^{\mathrm{DMD}}(x_{n+1},t_k) = 0$, we define the approximate boundary photocurrent at $t = t_k$ as

$$J_h^{\mathrm{DMD}}(x,t_k) = D_a \partial_x u_h^{\mathrm{DMD}}(x,t_k) = D_a \sum_{i=1}^{n} \tilde{u}_i(t_k) \partial_x v_i(x), \tag{5.2}$$

for $x = x_0$ and $x = x_{n+1}$. Note that owing to the local support of the basis functions $v_i(x)$, $J_h^{\mathrm{DMD}}(x_0, t_k)$ and $J_h^{\mathrm{DMD}}(x_0, t_k)$ only include contributions from the basis functions supported on elements κ_0 and κ_n, respectively. We then compare the predicted DMD flux with simulated experimental measurements of the current out of the device terminals. These measurements are obtained using the same numerical procedure as in Section 3, i.e., by computing a finite element solution $u_h^{\mathrm{FEM}}(x, t_k)$ and setting

$$J_h^{\mathrm{FEM}}(x_0, t_k) = D_a \partial_x u_h^{\mathrm{FEM}}(x_0, t_k) \tag{5.3}$$

$$J_h^{\mathrm{FEM}}(x_{n+1}, t_k) = D_a \partial_x u_h^{\mathrm{FEM}}(x_{n+1}, t_k). \tag{5.4}$$

5.1. Manufactured solution test. We first evaluate the ability of the DMD model to reproduce an artificially manufactured solution:

$$u^{\mathrm{MNF}}(x, t) := te^{-2t} \sin\big(\frac{3\pi x}{\ell}\big). \tag{5.5}$$

Observe that this solution satisfies homogenous initial and boundary conditions. Since (5.5) is expressed in closed-form, we can directly compute the boundary photocurrent of the manufactured solution:

$$J^{\mathrm{MNF}}(x, t) = D_a \partial_x u_h^{\mathrm{MNF}}(x, t) = D_a te^{-2t}\big(\frac{3\pi}{\ell}\big) \cos\big(\frac{3\pi x}{\ell}\big). \tag{5.6}$$

Thus, for this test we will compare J^{DMD} with the known manufactured solution current J^{MNF}, rather than with the simulated current J^{FEM} in (5.3)–(5.4). Substitution of $u^{\mathrm{MNF}}(x, t)$ into the governing equation (2.1) yields a generation density

$$g^{\mathrm{MNF}}(x, t) = \frac{\partial u^{\mathrm{MNF}}}{\partial t}(x, t) - D_a \frac{\partial^2 u^{\mathrm{MNF}}}{\partial x^2}(x, t) + \mu_a \mathbf{E}(x) \frac{\partial u^{\mathrm{MNF}}}{\partial x}(x, t) + \frac{1}{\tau_a} u^{\mathrm{MNF}}(x, t) \tag{5.7}$$

$$= (1 - 2t)e^{-2t} \sin\big(\frac{3\pi x}{\ell}\big) + D_a te^{-2t}\big(\frac{3\pi}{\ell}\big)^2 \sin\big(\frac{3\pi x}{\ell}\big) \tag{5.8}$$

$$+ \mu_a E(x) te^{-2t}\big(\frac{3\pi}{\ell}\big) \cos\big(\frac{3\pi x}{\ell}\big) + \frac{1}{\tau_a} te^{-2t} \sin\big(\frac{3\pi x}{\ell}\big) \tag{5.9}$$

such that when the ADE (2.1) is driven by $g^{\mathrm{MNF}}(x, t)$, its solution will exactly match the desired manufactured solution $u^{\mathrm{MNF}}(x, t)$. The inputs $\mathbf{g}_k$ to the DMD model are obtained by sampling $g^{\mathrm{MNF}}(x, t)$ according to the method described earlier.

Since the input $g^{\mathrm{MNF}}(x, t)$ corresponding to the manufactured solution is spatially irregular, a spatially uniform training input will generally result in poor performance. To address this, we design a sequence of localized pulses which will excite different regions of the device. This sequence will then be used as the training input for the DMD model. We choose to use a Gaussian profile that has been windowed by a cosine function as the spatial envelope for the input pulses, where the window function is applied to restrict the support of the envelope to a compact interval. This profile is consistent with experimentally viable radiation doses; other reasonable choices include *Lorenz* or *Voigt* profiles, which reflect different radiation broadening mechanisms. The windowed Gaussian profile with center x_i and support $[x_i - \frac{w}{2}, x_i + \frac{w}{2}]$ is given by

$$\rho_i(x) := \begin{cases} \cos\big(\pi \frac{x - x_i}{w}\big) \exp\big(-16\big(\frac{x - x_i}{w}\big)^2\big) & \text{if } x_i - \frac{w}{2} \le x \le x_i + \frac{w}{2} \\ 0 & \text{if } x < x_i - \frac{w}{2} \text{ or } x > x_i + \frac{w}{2} \end{cases} \tag{5.10}$$

where $x_i = i\frac{\ell}{N_{\text{pulses}}-1}$ for $i = 0, \ldots, N_{\text{pulses}} - 1$, and $w = \frac{\ell}{N_{\text{pulses}}-2}$. The profiles $\rho_i(x)$ are illustrated in Figure 5.1. Combining the spatial envelopes defined above with a square temporal envelope gives

$$g_i(x,t) = \begin{cases} \widehat{g}\rho_i(x) & \text{if } 0 \le t \le 0.5\,\mu\text{s} \\ 0 & \text{otherwise} \end{cases}, \tag{5.11}$$

where the value of $\widehat{g}$ is defined in (4.16). Now we can express the training input as

$$g_{\text{train}} = \sum_{i=0}^{N_{\text{pulses}}-1} g_i(x, t - t_i) \tag{5.12}$$

where $t_i = i(5.0\,\mu\text{s})$ for $i = 0, \ldots, N_{\text{pulses}} - 1$. For the manufactured solution (5.5), we choose $N_{\text{pulses}} = 10$.

To generate the training samples, we solve (2.1) with the source (5.12) and the parameters (4.16) on a mesh $\mathcal{X}^h$ comprising 512 uniform elements, and sample the solution in time at $\Delta t = 0.005\,\mu\text{s}$ increments.

Training input spatial envelopes

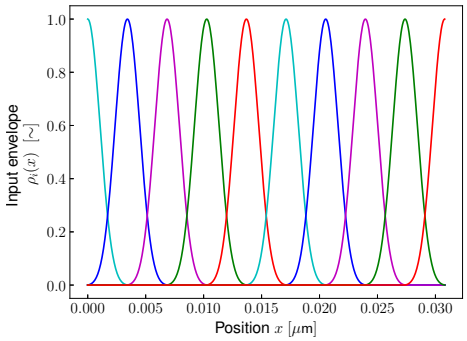


Fig. 5.1: Generation density function g_{train} used to obtain the training set, and g_{test} used to verify the model performance. The square edges in the training input are included to excite a wider range of dynamic modes, which is due to the high-bandwidth content in the sharp transitions.

Using the the manufactured photocurrent (5.6) and carrier density (5.5) and approximate photocurrent (5.2) and approximate carrier density (5.1) derived from the DMD model, we define the following error quantities:

$$E_J(t) := |J_h^{\text{DMD}}(t) - J^{\text{MNF}}(t)| \tag{5.13}$$

$$E_u(t) := |u_h^{\text{DMD}}(t) - u^{\text{MNF}}(t)| \tag{5.14}$$

which will characterize the ability of the DMD model to reproduce manually selected dynamic modes. For simplicity, in this work the electric field is not considered an input to the system, so the DMD model must be trained separately for each unique electric field strength.

We report results for the DMD model using several different values for the parameter p, which defines the number of the greatest singular values kept in the truncated SVD decomposition (4.8). Figure 5.3 compares the boundary photocurrent from the manufactured

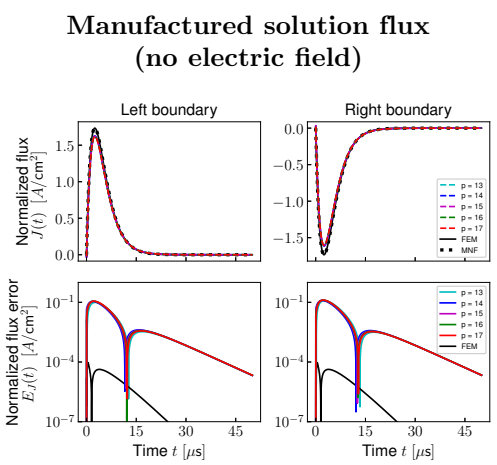


Fig. 5.2: Normalized photocurrent due to the manufactured input (5.7) with no electric field applied. The top two plots show the manufactured solution (5.6), FEM solution (5.3)-(5.4), and DMD solution (5.2), and the bottom two plots show the DMD and FEM error (5.13). The left two plots correspond to the flux out of the left side of the N-region, and similarly on the right.

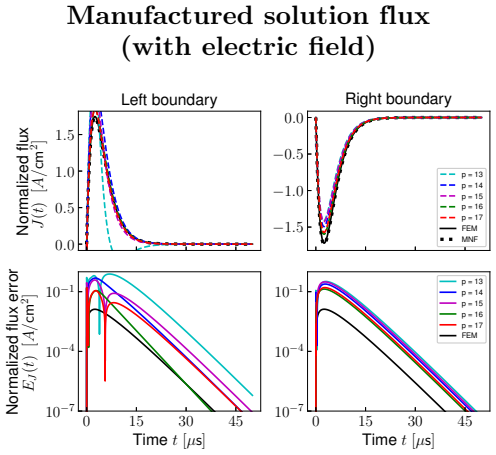


Fig. 5.3: Normalized photocurrent due to the manufactured input (5.7) with an electric field (4.16) applied. The top two plots show the manufactured solution (5.6), FEM solution (5.3)-(5.4), and DMD solution (5.2), and the bottom two plots show the DMD and FEM error (5.13). The left two plots correspond to the flux out of the left side of the N-region, and similarly on the right.

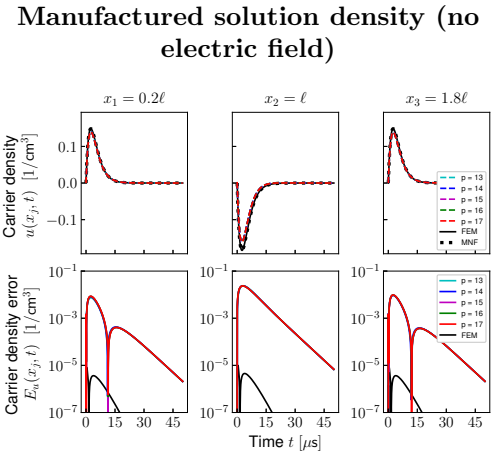


Fig. 5.4: Simulated excess carrier density due to the manufactured input (5.7) with no electric field applied. The top three plots show the manufactured solution (5.5), FEM solution (3.3), and DMD solution (5.1), and the bottom three plots show the DMD and FEM error (5.14).

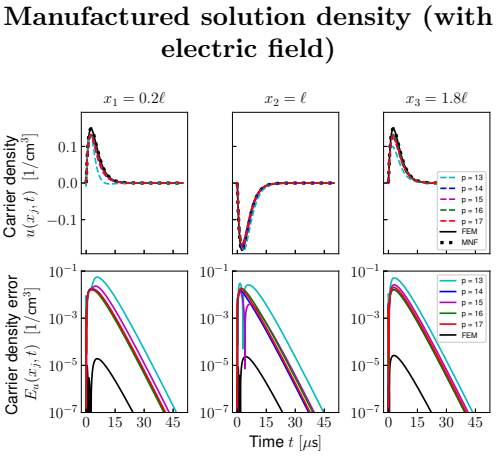


Fig. 5.5: Simulated excess carrier density due to the manufactured input (5.7) with an electric field (4.16) applied. The top three plots show the manufactured solution (5.6), FEM solution (3.3), and DMD solution (5.1), and the bottom three plots show the DMD and FEM error (5.14).

solution and from the FEM and DMD models subject to the manufactured input (5.7). Figures 5.5 and 5.7 show the carrier density from the manufactured solution and from the FEM and DMD models, with respect to time t or position x, respectively. Figures 5.2, 5.4, and 5.6 show the same, but in the absence of an electric field.

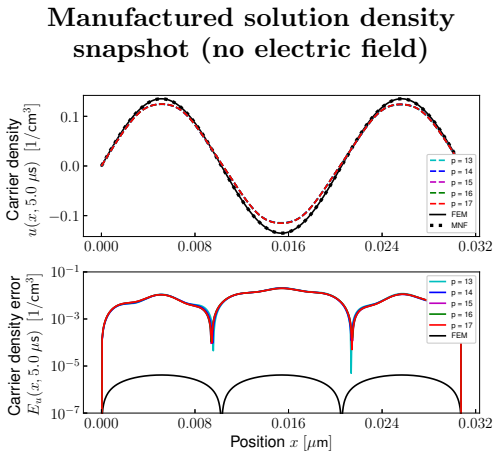


Fig. 5.6: Snapshot of the manufactured excess carrier density at $t = 5.0\,\mu s$ due to the manufactured input (5.7) with no electric field applied. The top plot shows the manufactured solution (5.6), FEM solution (3.3), and DMD solution (5.1), and the bottom plot shows the DMD and FEM error (5.14).

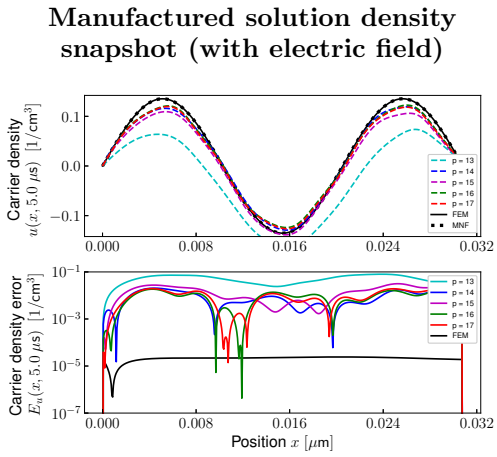


Fig. 5.7: Snapshot of the manufactured excess carrier density at $t = 5.0\,\mu s$ due to the manufactured input (5.7) with an electric field (4.16) applied. The top plot shows the manufactured solution (5.6), FEM solution (3.3), and DMD solution (5.1), and the bottom plot shows the DMD and FEM error (5.14).

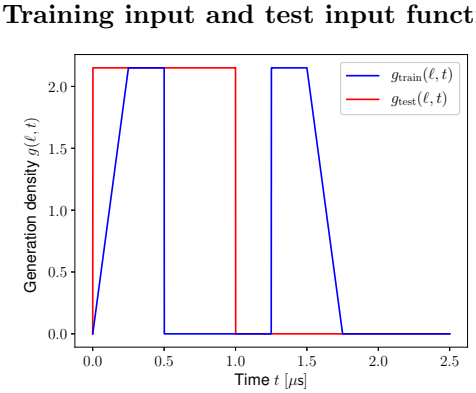


Fig. 5.8: Generation density function g_{train} used to obtain the training set, and g_{test} used to verify the model performance. The square edges in the training input are included to excite a wider range of dynamic modes, which is due to the high-bandwidth content in the sharp transitions.

5.2. Verification test. In this test we compare our compact DMD model with the Axness-Kerr [1] compact analytic model. We consider the case of a lightly doped diode as described in [1, Section B, p.2650] for which the length of the N-region equals 2 diffusion lengths (case $\xi_p = 2$ in [1, Figure 3, p.2651].) The device is irradiated by a 1.0 µs step pulse

$$g_{\text{test}}(x,t) = \begin{cases} \widehat{g} & \text{if } 0 \le t \le 1.0\,\mu s \\ 0 & \text{otherwise} \end{cases} \tag{5.15}$$

with the value of $\widehat{g}$ rescaled as in (4.16). This example from [1] corresponds to the parameters in 4.16 with the exception of a few corrections to account for typographical errors in that paper.

In contrast to the manufactured solution test where the desired input $g^{\mathrm{MNF}}(x,t)$ to the model is spatially irregular, now the target generation density, defined in (5.15), is spatially constant. As a result, the training input does not have to excite different regions of the device and can be chosen to be spatially constant as well. Thus, for this example we choose the *training input* $g_{\mathrm{train}}(x,t)$ to be a constant in space and a discontinuous piecewise linear in time function such that

$$g_{\mathrm{train}}(x,t) = \begin{cases} 0 & \text{if } 4t < 0\,\mu\text{s or } \ 2 \le 4t < 5\,\mu\text{s or } 4t \ge 7\,\mu\text{s} \\ \widehat{g}(4t) & \text{if } 0 \le 4t < 1\,\mu\text{s} \\ \widehat{g} & \text{if } 1 \le 4t < 2\,\mu\text{s or } 5 \le 4t < 6\,\mu\text{s} \\ \widehat{g}(7-4t) & \text{if } 6 \le 4t < 7\,\mu\text{s} \end{cases}, \tag{5.16}$$

See Figure 5.8 for an illustration of g_{train} and g_{test}.

We then solve (2.1) with the source (5.16) and the parameters (4.16) on a mesh $\mathcal{X}^h$ comprising 1024 uniform elements, and sample the solution in time at $\Delta t = 0.0025\,\mu\text{s}$ increments. The compact DMD photocurrent model for the device is now defined according to (4.11).

Remark 3. *Since the input is applied uniformly across the entire device (and thus the entire state space), we could reduce the input dimension of the DMD model (4.11) from N to 1, however the higher dimensional input permits spatially irregular excitations such as localized radiation pulses or non-transversal plane waves as required for the manufactured solution test in Section 5.1.*

To determine an appropriate dimension for the reduced-order DMD model, it is informative to inspect the relative magnitudes of the singular values from the decompositions of the sample matrices. For the training input (5.16), Figure 5.9 illustrates the magnitude roll-off in the singular values for the state and state-input sample matrix decompositions. Observe that even for tight error thresholds, only a few modes are necessary to construct an accurate approximation of the state transition and input matrices.

We wish to emphasize that including too many modes in the reduced-order model often leads to a realization which is unstable over a long time horizon. This phenomenon occurs due to unstable modes corresponding to small singular values. Including these low-magnitude modes will typically lead to a better fit for the state dynamics resulting from the training input over the original training time horizon, but may lead to model divergence over a longer time horizon, signifying an example of *overfitting*.

Based on the photocurrent (5.3)-(5.4) and carrier density (3.3) derived from the finite-elements model and the approximate photocurrent (5.2) and approximate carrier density (5.1) derived from the DMD model, we redefine the following error quantities:

$$E_J(t) := |J_h^{\mathrm{DMD}}(t) - J_h^{\mathrm{FEM}}(t)| \tag{5.17}$$

$$E_u(t) := |u_h^{\mathrm{DMD}}(t) - u_h^{\mathrm{FEM}}(t)| \tag{5.18}$$

which will facilitate the performance evaluation of the DMD model for typical, spatially uniform input functions. As before, the DMD model must be trained separately for each electric field strength.

Again, we show the results for the DMD model using multiple values for the parameter p. However, observe that for the case where the generation density is spatially uniform, fewer modes are necessary to achieve good performance than when the input magnitude varies along the length of the device, as in the manufactured solution test. Figures 5.10

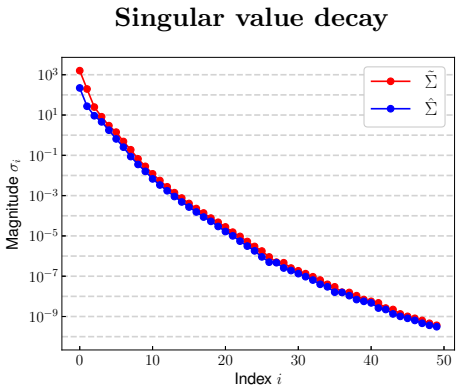


Fig. 5.9: When listed from greatest to least, the singular values in the decomposition of the state-input sample matrix S ($\tilde{\Sigma}$) and state sample matrix X' ($\hat{\Sigma}$) demonstrate a nearly exponential decay in magnitude.

and 5.11 compare the boundary photocurrent produced by the FEM and DMD models for the training input (5.16) and test input (5.15), respectively. Figures 5.12 and 5.13 show the same, but in the absence of an electric field. Figures 5.14 and 5.15 illustrate the simulated excess carrier density (i.e., the internal state) from the FEM and DMD models for both the training and test inputs.

Several conclusions can be drawn from these results. First, the photocurrent plots at the left boundary of the N-region shown in Figures 5.11 and 5.13 are in an excellent agreement with the results reported in [1, Figure 3, p.2651] for $\xi_p = 2$. Second, the error plots on the bottom rows of the figures quantify the differences between the reference FEM solution and its DMD approximation as a function of the number p of selected dynamical modes. These results reveal that, as expected, the error decreases with increase of the number of dynamic modes; however, even with just 6 modes selected the DMD photocurrent model yields excellent accuracy. Overall, these results suggest that a data-driven approach is indeed a viable and effective alternative to traditional analytic model development that can be used to quickly develop accurate and computationally efficient photocurrent models directly from data.

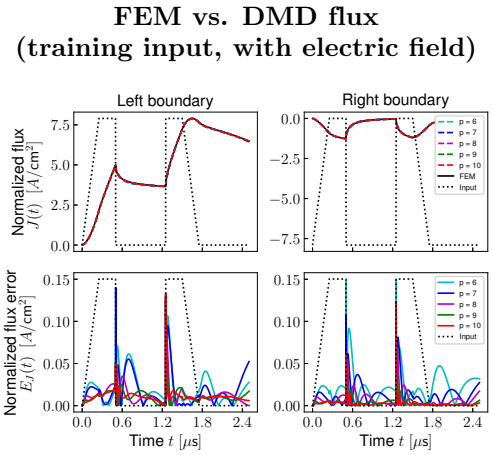


Fig. 5.10: Normalized photocurrent due to the training input (5.16) with an electric field (4.16) applied. The top two plots show the FEM solution (5.3)-(5.4) and DMD solution (5.2), and the bottom two plots show the DMD training error (5.17). The left two plots correspond to the flux out of the left side of the N-region, and similarly on the right.

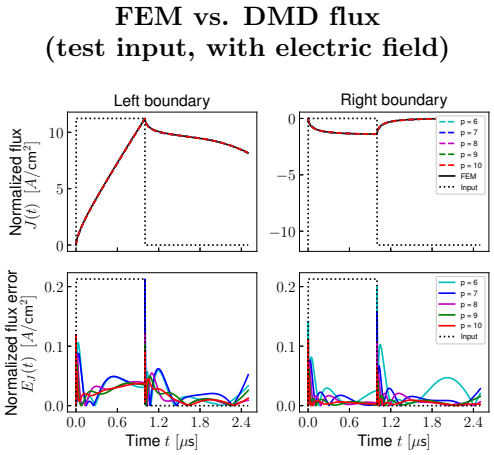


Fig. 5.11: Normalized photocurrent due to the test input (5.15) with an electric field (4.16) applied. The top two plots show the FEM solution (5.3)-(5.4) and DMD solution (5.2), and the bottom two plots show the DMD training error (5.17). The left two plots correspond to the flux out of the left side of the N-region, and similarly on the right.

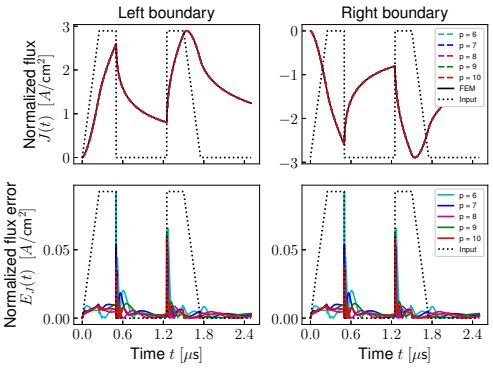


Fig. 5.12: Normalized photocurrent due to the training input (5.16) with no electric field applied. The top two plots show the FEM solution (5.3)-(5.4) and DMD solution (5.2), and the bottom two plots show the DMD training error (5.17). The left two plots correspond to the flux out of the left side of the N-region, and similarly on the right.

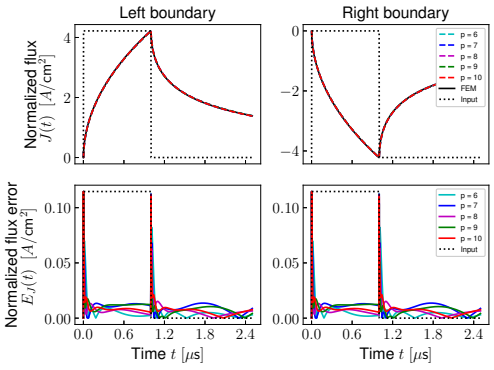


Fig. 5.13: Normalized photocurrent due to the test input (5.15) with no electric field applied. The top two plots show the FEM solution (5.3)-(5.4) and DMD solution (5.2), and the bottom two plots show the DMD training error (5.17). The left two plots correspond to the photocurrent out of the left side of the N-region, and similarly on the right.

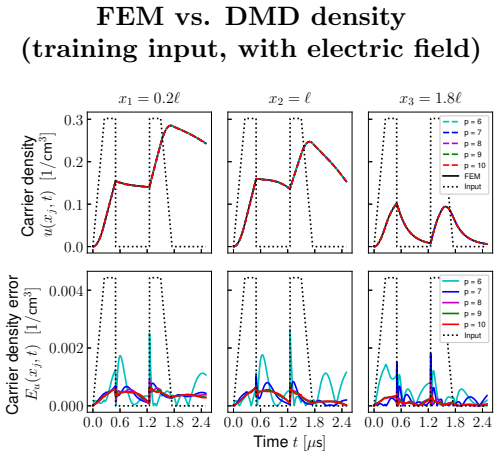


Fig. 5.14: Simulated excess carrier density due to the training input (5.16) with an electric field (4.16) applied. The top three plots show the FEM solution (3.3) and DMD solution (5.1), and the bottom three plots show the DMD training error (5.18).

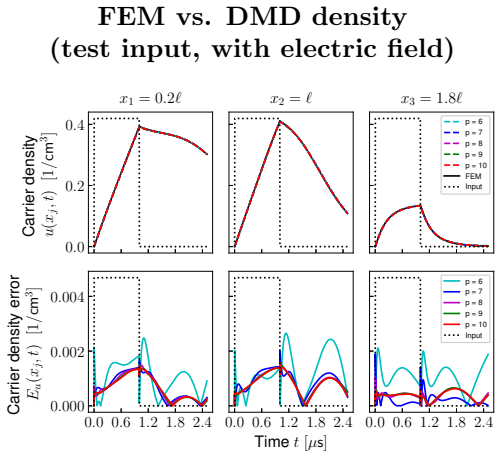


Fig. 5.15: Simulated excess carrier density due to the test input (5.15) with an electric field (4.16) applied. The top three plots show the FEM solution (3.3) and DMD solution (5.1), and the bottom three plots show the DMD test error (5.18).

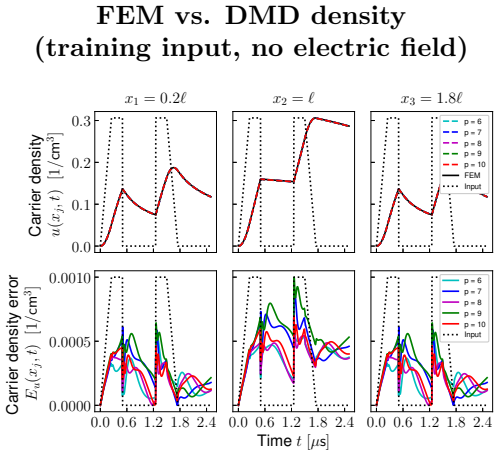


Fig. 5.16: Simulated excess carrier density due to the training input (5.16) with no electric field applied. The top three plots show the FEM solution (3.3) and DMD solution (5.1), and the bottom three plots show the DMD training error (5.18).

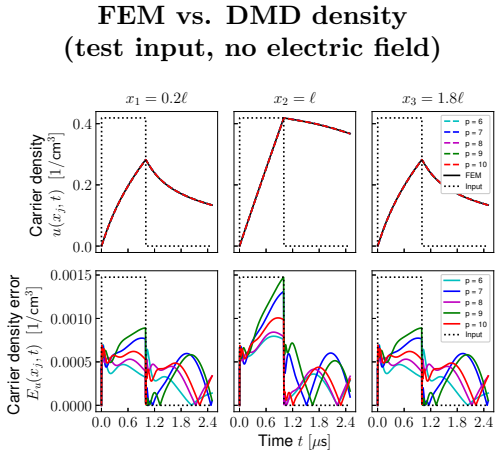


Fig. 5.17: Simulated excess carrier density due to the test input (5.15) with no electric field applied. The top three plots show the FEM solution (3.3) and DMD solution (5.1), and the bottom three plots show the DMD test error (5.18).

6. Conclusions. We have developed a compact data-driven delayed photocurrent model given by a low-dimensional discrete-time dynamical system, which approximates the flow map of the Ambipolar Diffusion Equation. To obtain the approximate flow map we use the Ambipolar Diffusion Equation to reconstruct numerically the internal state of the device, which is not directly observable through a laboratory measurement, and then apply Dynamic Mode Decomposition to the simulated internal state samples. In doing so physics knowledge is incorporated into the model development, which allows us to obtain models from *sparse data sets* that accurately approximate the dynamics of the excess carrier density. This in turn allows us to accurately estimate the induced current at the device boundaries, which is the quantity required for circuit simulations.

Our results confirm that such physics-aware data-driven models are a viable alternative to traditional analytic compact models that use simplified analytic solutions of the governing equations and often must undergo recalibration and/or redevelopment to include new physics effects.

Our future work will consider extension of the approach to include an additional parameter identification step, and to the fully coupled DDE (1.1)-(1.3). The latter will allow us to model the total photocurrent in the device and eliminate the need to split it into three separate regions.

Applying nonlinear observable functions to the state of the DDE would allow us to model the nonlinear problem using the same DMD algorithm, requiring little to no additional computational cost for running and training the DMD model (besides applying the nonlinearities to the measurement data, which is inexpensive). We also plan to incorporate and test our models in circuit simulators to demonstrate their utility for circuit design and analysis tasks.

Acknowledgments. This work was supported by the Sandia National Laboratories (SNL) Laboratory-directed Research and Development (LDRD) program, and the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research under Award Number DE-SC-0000230927 and under the Collaboratory on Mathematics and Physics-Informed Learning Machines for Multiscale and Multiphysics Problems (PhILMs) project. The work of J. Hanson was performed as part of Sandia Summer Student Program Internship at the Computer Science Research Institute (CSRI).

This work benefited from numerous discussions and interactions with our colleagues Eric Keiter, Suzey Gao and Larry Musson who shared their expertise with compact model development and TCAD simulations. We are grateful for their help and advice throughout the preparation of this paper.

REFERENCES

[1] C. L. Axness, B. Kerr, and T. F. Wunsch, *Analytic light—or radiation—induced pn junction photocurrent solutions to the multidimensional ambipolar diffusion equation*, Journal of Applied Physics, 96 (2004), pp. 2646–2655.

[2] Y. Bar-Sinai, S. Hoyer, J. Hickey, and M. P. Brenner, *Learning data-driven discretizations for partial differential equations*, Proceedings of the National Academy of Sciences, 116 (2019), pp. 15344–15349.

[3] P. Blonigan, K. Carlberg, F. Rizzi, M. Howard, and J. Fike, *Model reduction for hypersonic aerodynamics via conservative lspg projection and hyper-reduction*, AIAA Paper 2020-0104, AIAA Scitech 2020 Forum, Orlando, FL, (2020).

[4] P. Ciarlet, *The Finite Element Method for Elliptic Problems*, SIAM Classics in Applied Mathematics, SIAM, Philadelphia, 2002.

[5] E. W. Enlow and D. R. Alexander, *Photocurrent modeling of modern microcircuit pn junctions*, IEEE Transactions on Nuclear Science, 35 (1988), pp. 1467–1474.

[6] T. A. Fjeldly, Y. Deng, M. S. Shur, H. P. Hjalmarson, A. Muyshondt, and T. Ytterdal, *Modeling of high-dose-rate transient ionizing radiation effects in bipolar devices*, IEEE Transactions on Nuclear Science, 48 (2001), pp. 1721–1730.

[7] J. H. Tu, C. W. Rowley, D. M. Luchtenburg, S. L. Brunton, and J. Nathan Kutz, *On dynamic mode decomposition: Theory and applications*, Journal of Computational Dynamics, 1 (2014), pp. 391–421.

[8] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, CoRR, abs/1512.03385 (2015).

[9] B. Kerr, C. L. Axness, J. C. Verley, C. E. Hembree, and E. R. Keiter, *A new time-dependent analytic model for radiation-induced photocurrent in finite 1d epitaxial diodes*, Sandia Report SAND2012-2161, Sandia National Laboratories, 2012.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *Imagenet classification with deep convolutional neural networks*, Commun. ACM, 60 (2017), pp. 84–90.

[11] Y. LeCun, Y. Bengio, and G. Hinton, *Deep learning*, Nature, 521 (2015), pp. 436 EP –.

[12] J. L. Proctor, S. L. Brunton, and J. N. Kutz, *Dynamic mode decomposition with control*, SIAM Journal on Applied Dynamical Systems, 15 (2016), pp. 142–161.

[13] C. Rackauckas, Y. Ma, J. Martensen, C. Warner, K. Zubov, R. Supekar, D. Skinner, and A. Ramadhan, *Universal differential equations for scientific machine learning*, 2020.

[14] M. Raissi, P. Perdikaris, and G. Karniadakis, *Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations*, Journal of Computational Physics, 378 (2019), pp. 686 – 707.

[15] G. Rozza, D. B. P. Huynh, and A. T. Patera, *Reduced basis approximation and a posteriori error estimation for affinely parametrized elliptic coercive partial differential equations*, Archives of Computational Methods in Engineering, 15 (2008), p. 229.

[16] P. J. SCHMID, *Dynamic mode decomposition of numerical and experimental data*, Journal of Fluid Mechanics, 656 (2010), pp. 5–28.

[17] S. Selberherr, *Analysis and simulation of semiconductor devices*, Springer-Verlag, Berlin, 1984.

[18] L. F. Shampine and M. W. Reichelt, *The matlab ode suite*, SIAM Journal on Scientific Computing, 18 (1997), pp. 1–22.

[19] J. L. Wirth and S. C. Rogers, *The transient response of transistors and diodes to ionizing radiation*, IEEE Transactions on Nuclear Science, 11 (1964), pp. 24–38.

[20] T. F. Wunsch and C. L. Axness, *Modeling the time-dependent transient radiation response of semiconductor junctions*, IEEE Transactions on Nuclear Science, 39 (1992), pp. 2158–2169.

CLASSICAL APPROXIMATIONS OF SPECIAL FERMIONIC MANY-BODY SYSTEMS

DANIEL G. HOTHEM[‡] AND OJAS PAREKH[§]

Abstract. Finding the lowest energy or ground states of quantum many-body systems with local interactions is a topic of study in a diverse range of fields such as condensed matter physics, quantum chemistry, and theoretical computer science. Difficult to accomplish in general, there has been a recent growth of interest in rigorous classical approximations to ground states of such systems. Our paper builds upon the work of Bravyi, Gosset, Koenig, and Temme to better understand how approximation schemes for k-local qubit systems can be applied to fermionic many-body systems with local interactions. We accomplish this by applying Gharibian and Parekh's product state approach to a subclass of fermionic Hamiltonians.

1. Introduction. Quantum many-body problems with local interactions are of interest to a diverse class of researchers ranging from condensed matter physicists to theoretical computer scientists. Initially posed to study various physical systems, these quantum many-body systems are a natural setting for the quantum analog of many classical constraint satisfaction problems through their connection to k-local Hamiltonians. Past work, such as that in [2] and [3], has shown how the celebrated classical algorithm of Goemans and Williamson[4] can be applied to optimization problems involving k-local Hamiltonians.

In this paper we begin by reviewing the basics of two-local qubit Hamiltonians in Section 2 by following Gharibian and Parekh's paper[3]. In Section 3, we outline the approach taken by Bravyi et al.[2] to approximating traceless two-local fermionic Hamiltonians. We then end in Section 4 by demonstrating that the product states used by Gharibian and Parekh form a special subclass of the states used by Bravyi et al., uniquely characterized by a diagonalizability criterion on the state's covariance matrix. This further adds to our understanding of the connection between k-local systems of qubits and the k-local systems of fermionic particles that are so widely found across modern physics.

2. Two-Local Qubit Hamiltonians. Let $N \in \mathbb{Z}_{>0}$ be the number of qubits in a system. Any traceless 2-local qubit Hamiltonian may be written as

$$H = H_1 + H_2 \tag{2.1}$$

$$H_1 = \sum_{j=1}^{3N} D_j P_j \text{ and } H_2 = \sum_{i,j=1}^{3N} D_{ij} P_i P_j \tag{2.2}$$

Here each P_j represents a Pauli operator. Specifically, the Pauli operator acting on the j-th qubit is given by $P_{3j-2} = X_j$, $P_{3j-1} = Y_j$, and $P_{3j} = Z_j$ where

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}.$$

Typically, one is interested in finding the maximum eigenvalue $\lambda_{max}(H)$ of H. Unfortunately, computing this exactly or to within some small additive error is QMA-hard [5]. As detailed earlier, it is often more convenient to approximate $\lambda_{max}(H)$ by some $\tilde{\lambda}$ with a

[‡]Department of Mathematics, University of Oregon, dhothem@uoregon.edu
[§]Sandia National Laboratories, odparek@sandia.gov

good *approximation ratio* $\tilde{\lambda}/\lambda_{max}(H)$. In order to accomplish this we seek to maximize the expectation of H over a particular subset of quantum states.

A natural subset to optimize over is the subset of product states[3]. For N qubits, a product state is an N-fold tensor product of single-qubit density matrices. The set of single-qubit density matrices is denoted $\mathcal{D}(\mathbb{C}^2)$. The density matrix of a single qubit may be written as

$$\rho = \frac{1}{2}\Big(I + \alpha X + \beta Y + \gamma Z\Big) = \frac{1}{2}(I + \boldsymbol{r}\cdot\boldsymbol{\sigma}) \text{ where } \|\boldsymbol{r}\| \leq 1 \text{ and } \boldsymbol{\sigma} = (X, Y, Z).$$

Thus any N-qubit product state can be written as

$$\rho = \frac{1}{2^N}\bigotimes_{i=1}^{N}\Big(I + \alpha_i X_i + \beta_i Y_i + \gamma_i Z_i\Big). \tag{2.3}$$

More generally, any separable state on N qubits is an element of the convex hull of the set of product states, SEP $= \text{conv}(\otimes_{i=1}^{N}\rho_i | \rho_i \in \mathcal{D}(\mathbb{C}^2))$. Our optimization problem is then recast as

$$\lambda_{\text{prod}} = \max_{\rho\in\text{SEP}} \text{tr}(H\rho).$$

Because of convexity, λ_{prod} is obtained by some pure product state ρ. Pure states are those for which $\text{tr}(\rho^2) = 1$. They are those for which there is no classical uncertainty about the state of the system, although a natural quantum mechanical uncertainty still remains.

3. Traceless Two-Local Fermionic Hamiltonians. Product states are not the only convenient states to optimize over. Another such set of states are Gaussian states, which come from viewing a two-local qubit Hamiltonian as a traceless two-local fermionic Hamiltonian. Following [2], The Hilbert space of n fermionic modes has a basis given by binary strings of length n, $x \in \{0,1\}^n$. A one in the j-th slot means that the j-th mode is occupied, while a zero in the j-th slot means that the mode is unoccupied.

Hamiltonians for these systems are built out of annihilation and creation operators. The j-th annihilation operator a_j reduces a state's j-th mode's occupation number by one if the j-th mode is occupied or maps the state to 0 if the j-th mode is unoccupied. The j-th creation operator $a_j^\dagger$ increases the state's j-th occupation number if the j-th mode is unoccupied and otherwise maps the state to 0. Crucially, the annihilation and creation operators satisfy the following canonical anticommutation relations:

$$\{a_i, a_j\} = 0 \text{ and } \{a_i^\dagger, a_j\} = \delta_{ij} I.$$

Typical Hamiltonians of interest take the following form

$$H = H_1 + H_2 + \omega I \tag{3.1}$$

$$H_1 = \sum_{p,q=1}^{n} V_{p,q} a_p^\dagger a_q \text{ and } H_2 = \sum_{p,q,r,s=1}^{n} W_{pqrs} a_p^\dagger a_q^\dagger a_r a_s \tag{3.2}$$

The coefficients $V_{p,q}$ and $W_{p,q,r,s}$ are complex numbers carefully chosen so that H is hermitian. Any such Hamiltonian can be rewritten as a two-local Hamiltonian as laid out in [2].

Fermionic Gaussian states are more easily defined via Majorana operators rather than annihilation and creation operators. To each fermionic mode j, one can associate two Majorana operators or modes by

$$c_{2j-1} = a_j + a_j^\dagger \text{ and } c_{2p} = (-i)(a_p - a_p^\dagger). \tag{3.3}$$

By definition, each Majorana operator is hermitian. Furthermore they satisfy the commutation rule $\{c_p, c_q\} = 2\delta_{p,q}I$. They generate the Clifford algebra C_{2n}. Gaussian states correspond to a subset of C_{2n} [1]:

DEFINITION 3.1. *Let $\mathcal{H}$ be a Hilbert space of n fermionic modes. For any orthogonal matrix $R \in O(2n)$ define the following unitary operator $U_R \in U(n)$ by*

$$(U_R)^\dagger c_p (U_R) = \sum_{q=1}^{2n} R_{p,q} c_q. \tag{3.4}$$

A state $\psi \in \mathcal{H}$ is Gaussian if it is of the form $U_R|0\rangle^n$ for some $R \in O(2n)$.

For our purposes it is easier to work with mixed Gaussian states [1]. These are best defined with the use of Grassmann variables. Any two Grassmann variables anti-commute $\theta_i\theta_j = -\theta_j\theta_i$, forcing each variable to square to zero. Together $2n$ Grassmann variables generate the Grassmann algebra $\mathcal{G}_{2n}$. There exists an isomorphism of *vector spaces* between C_{2n} and $\mathcal{G}_{2n}$ which maps monomials of Majorana operators to

$$\omega(c_{j_1} \dots c_{j_k}, \theta) = \theta_{j_1} \dots \theta_{j_k}.$$

DEFINITION 3.2. *Let $\rho \in C_{2n}$ be the density operator for some state on n fermionic modes. The state is Gaussian if and only if the Grassmann representation of ρ has the following form*

$$\omega(\rho, \theta) = \frac{1}{2^n} \exp(\frac{i}{2}\theta^T M \theta)$$

for some real $2n \times 2n$ antisymmetric matrix M. M is called the correlation matrix of ρ.

Note that if $\rho \in C_{2n}$ is Gaussian, then its correlation matrix stores information about the two-point correlators of ρ. In fact, each entry of M is given by

$$M_{ab} = \frac{i}{2}\text{tr}(\rho[c_a, c_b]) = \begin{cases} \text{tr}(i\rho c_a c_b) \text{ for } a \neq b \\ 0 \text{ for } a = b \end{cases} \tag{3.5}$$

A quick computation verifies that if U_R is defined as in equation 3.4, then

$$\omega((U_R)^\dagger \rho(U_R), \theta) = \omega(\rho, \eta) \text{ where } \eta_j = \sum_{b=1}^{2n} R_{ab} c_b. \tag{3.6}$$

Under this basis change, the correlation matrix of $(U_R)^\dagger \rho(U_R)$ becomes $R^T M R$.

Not only are Majorana operators useful for defining Gaussian states, but by inverting the equations in equation 2.3 we can rewrite any traceless two-local fermionic Hamiltonian from equations 2.1 and 2.2 as below[2]:

$$H = H_1 + H_2 \tag{3.7}$$

$$H_1 = \sum_{p,q=1}^{2n} iv_{pq}c_q c_q \text{ and } H_2 = \sum_{p,q,r,s=1}^{2n} W_{pqrs}c_p c_q c_r c_s \tag{3.8}$$

These Majorana operators also allow one to embed any qubit Hamiltonian of the form in equation 2.1 as a traceless two-local fermionic Hamiltonian [2]. Let $n = 3N/2$ and assume for simplicity that N is even. Map the Paulis into quadratic polynomials of the Majorana operators as follows:

$$X_p = ic_{3p-2}c_{3p-1}, \quad Y_p = ic_{3p-1}c_{3p}, \quad Z_p = ic_{3p-2}c_{3p}. \tag{3.9}$$

This transformation preserves eigenvalues. In order to preserve the proper normalization of a mixed state you must rescale the pushforward of any qubit density matrix by a factor of $\frac{1}{2^{N/2}}$.

4. Product States as Gaussian States. The embedding of the Pauli operators into C_{2n} leads to a natural question. Are product states Gaussian? The following lemma answers this question affirmatively.

LEMMA 4.1. *Let ρ be the density matrix for some product state on N-qubits. Then ρ embeds as a Gaussian state under the embedding in equation 3.9. Furthermore, any Gaussian state whose correlation matrix M can be brought into the following 3x3 anti-symmetric block diagonal form by some matrix $R \in SO(3N)$ comes from some embedding of a product state*

$$R^T M R = \frac{1}{2^{3N/2}} \operatorname{diag}\left(\begin{bmatrix} 0 & \alpha_1 & \gamma_1 \\ -\alpha_1 & 0 & \beta_1 \\ -\gamma_1 & -\beta_1 & 0 \end{bmatrix}, \ldots, \begin{bmatrix} 0 & \alpha_N & \gamma_N \\ -\alpha_N & 0 & \beta_N \\ -\gamma_N & -\beta_N & 0 \end{bmatrix} \right). \tag{4.1}$$

It is then natural to ask what elements of $R \in SO(3N)$ transform the correlation matrices of qubit Gaussian states into their canonical 2x2 block diagonal form. In this case, each 3x3 block can be handled separately. In this case, $R \in SO(3)$. The columns of R correspond to the two singular vectors of M as well as the one zero eigenvector of M. Details are left to the appendix.

5. Conclusions. In this paper we further explored the connection established by Bravyi, Gosset, Koenig, and Temme between two-local qubit Hamiltonians and traceless two-local fermionic Hamiltonians. Specifically, we demonstrated that the class of product state solutions to a qubit Hamiltonian make up a subclass of fermion Gaussian states. We further classified all fermionic Gaussian states that come from product states of qubits. Fruitful future work includes classifying traceless two-local fermionic Hamiltonians that come from two-local qubit Hamiltonians as well as investigating other nice subclasses of fermionic Gaussian states.

Appendix A. Proof of Lemma 4.1. Assume that the number of qubits N is even. We begin by showing that the pushforward of the density matrix of a product state to C_{3N} fulfills definition 3.2. Let ρ be the density matrix of a product state as defined in equation 2.3. When appropriately scaled, ρ maps to the following polynomial in C_{3N} under the embedding described in equation 3.9

$$\rho \to \tilde{\rho} = \frac{1}{2^{3N/2}} \prod_{j=1}^{N} \Big(I + i\alpha_j c_{3j-2}c_{3j-1} + i\beta j c_{3j-1}c_{3j} + i\gamma_j c_{3j-2}c3j \Big). \tag{5.1}$$

Expanding out $\tilde{\rho}$ gives

$$\tilde{\rho} = \frac{1}{2^{3N/2}} \Big(I + i \sum_{j=1}^{N} (\alpha_j c_{3j-2}c_{3j-1} + \beta_j c_{3j-1}c_{3j} + \gamma_j c_{3j-2}c_{3j}) + O(c^4) \Big). \tag{5.2}$$

The higher order terms in the expansion of $\tilde{\rho}$ do not matter in the computation of $\tilde{\rho}$'s correlation matrix M (any monomial of Majorana operators that does not resolve to the identity is traceless). Because distinct Majorana operators anti-commute, $\tilde{\rho}$ must have a correlation matrix of the form in equation 4.1.

Using this correlation matrix M a short computation verifies that

$$\theta^T M\theta = 2\sum_{j=1}^{N} \alpha_j\theta_{3j-2}\theta_{3j-1} + \beta_j\theta_{3j-1}\theta_{3j} + \gamma_j\theta_{3j-2}\theta_{3j} \tag{5.3}$$

Substituting this in to the definition of $\omega(\tilde{\rho},\theta)$ and using the fact that each summand is in the center of the Grassmann algebra gives

$$\omega(\tilde{\rho},\theta) = \frac{1}{2^{3N/2}}\prod_{j=1}^{N} \exp\Big(i(\alpha_j\theta_{3j-2}\theta_{3j-1} + \beta_j\theta_{3j-1}\theta_{3j} + \gamma_j\theta_{3j-2}\theta_{3j})\Big) \tag{5.4}$$

$$= \frac{1}{2^{3N/2}}\prod_{j=1}^{N} \Big(I + i\alpha_j\theta_{3j-2}\theta_{3j-1} + i\beta_j\theta_{3j-1}\theta_{3j} + i\gamma_j\theta_{3j-2}\theta_{3j}\Big). \tag{5.5}$$

This is exactly what one gets for $\omega(\tilde{\rho},\theta)$ by using equation 5.1 directly.

Now suppose that ρ is a Gaussian state whose correlation matrix M is 3x3 block diagonalized as in equation 4.1 by some element $R \in SO(3N)$. Let ϕ be the embedding laid out in equation 3.9 and define a new embedding $\tilde{\phi}$ by

$$\begin{aligned} X_p &= i\sum_{b=1}^{3N} R_{3p-2,b}c_b \sum_{b=1}^{3N} R_{3p-1,b}c_b, \quad Y_p = i\sum_{b=1}^{3N} R_{3p-1,b}c_b \sum_{b=1}^{3N} R_{3p,b}c_b, \\ Z_p &= i\sum_{b=1}^{3N} R_{3p-2,b}c_b \sum_{b=1}^{3N} R_{3p,b}c_b \end{aligned}. \tag{5.6}$$

It is established in [1] that $(U_R)^\dagger\rho(U_R)$ has R^TMR as its correlation matrix. From the correlation matrix, we have

$$(U_R)^\dagger\rho(U_R) = \phi\Big(\frac{1}{2^N}\prod_{j=1}^{N} I + \alpha_j X_j + \beta_j Y_j + \gamma_j Z_j\Big).$$

Let $F \in \mathbb{C}[x_1,\ldots,x_{3N}]$ be the polynomial that, when evaluated on the Majorana operators, gives $(U_R)^\dagger\rho(U_R)$. By using equation 3.4 to commute $(U_R)^\dagger$ past the Majorana operators, we get that

$$\rho = F\Big(\sum_{b=1}^{3N} R_{1b}c_b, \ldots, \sum_{b=1}^{3N} R_{3N,b}c_b\Big) = \tilde{\phi}\Big(\frac{1}{2^N}\prod_{j=1}^{N} I + \alpha_j X_j + \beta_j Y_j + \gamma_j Z_j\Big).$$

Therefore ρ comes from some embedding of a product state. ■

Appendix B. Some elements of $SO(3)$. Let M be a real anti-symmetric 3x3 matrix of the following form:

$$M = \begin{bmatrix} 0 & \alpha & \gamma \\ -\alpha & 0 & \beta \\ -\gamma & -\beta & 0 \end{bmatrix}$$

Without loss of generality, assume that $\beta \neq 0$. M is brought into canonical form by the following element of $SO(3)$

$$\begin{bmatrix} \frac{\alpha}{\sqrt{\alpha^2+\beta^2+\gamma^2}} & \frac{\gamma}{\beta} & -\frac{-\alpha\beta\left(-\beta^2\gamma^2+\left(\alpha^2+\beta^2\right)\left(\alpha^2+\gamma^2\right)\right)-\beta\gamma\left(\alpha\beta^2\gamma+\alpha\gamma\left(\alpha^2+\gamma^2\right)\right)}{(\alpha^2+\gamma^2)(-\beta^2\gamma^2+(\alpha^2+\beta^2)(\alpha^2+\gamma^2))} \\ -\frac{\alpha\gamma}{\beta\sqrt{\alpha^2+\beta^2+\gamma^2}} & 1 & -\frac{\alpha\beta^2\gamma+\alpha\gamma\left(\alpha^2+\gamma^2\right)}{-\beta^2\gamma^2+(\alpha^2+\beta^2)(\alpha^2+\gamma^2)} \\ -\frac{\beta}{\sqrt{\alpha^2+\beta^2+\gamma^2}} & -\frac{\gamma^2}{\beta\sqrt{\alpha^2+\beta^2+\gamma^2}} & 0 \qquad\qquad 1 \end{bmatrix}$$

The first two column vectors correspond the singular values of M, while the final column vector is a zero eigenvector of M.

REFERENCES

[1] S. Bravyi, *Lagrangian representation for fermionic linear optics*, 2004.

[2] S. Bravyi, D. Gosset, R. König, and K. Temme, *Approximation algorithms for quantum many-body problems*, Journal of Mathematical Physics, 60 (2019), p. 032203.

[3] S. Gharibian and O. Parekh, *Almost optimal classical approximation algorithms for a quantum generalization of max-cut*, Proceedings of the 22nd International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX), 145 (2019), pp. 31:1–31:17.

[4] M. Goemans and D. Williamson, *Improved approximation algorithsm for maximum cut and satisfiability problems using semidefinite programming*, Journal of the JACM, 42 (1995).

[5] A. Y. Kitaev, A. H. Shen, and M. N. Vyalyi, *Classical and Quantum Computation*, American Mathematical Society, 2002.

ETD-RK EXPONENTIAL INTEGRATORS FOR THE NONHYDROSTATIC ATMOSPHERE MODEL HOMME-NH.

CASSIDY F. KRAUSE* AND ANDREW J. STEYER†

Abstract. The nonhydrostatic atmosphere model HOMME-NH requires integrating an additively partitioned stiff initial value problem. The current standard is to integrate in time using a horizontally explicit, vertically implicit (HEVI) partitioning with an implicit-explicit Runge Kutta (IMEX-RK) scheme. However, the linearization of the stiff terms in HOMME-NH yields a sparse matrix consisting of blocks that are either diagonal or tridiagonal, making it an excellent candidate for efficient computation of the matrix exponential. Here we explore exponential time differencing Runge Kutta (ETD-RK) methods as an alternative time integrator for HOMME-NH. We present various ETD-RK methods that maximize efficiency by keeping stage value computations and storage to a minimum, and we compare the accuracy and stability regions of these methods.

1. Introduction. Exponential time differencing-Runge Kutta (ETD-RK) methods were developed first in the 1970s as time integrators to solve stiff initial value problems (IVPs), but the computational expense of forming the matrix exponential, a key component of ETD-RK methods, prevented their widespread use until more recent decades. We consider the use of ETD-RK methods with the nonhydrostatic atmosphere model HOMME-NH.

Once discretized in space, the evolution of the state variables of HOMME-NH can be written as an ODE that is additively partitioned into stiff and nonstiff terms. The current implementation of HOMME-NH uses an implicit-explicit Runge Kutta (IMEX-RK) time integrator, but there are many other timestepping techniques that also work well with stiff IVPs, such as ETD-RK methods. Furthermore, the linearization of the stiff terms of HOMME-NH provides a Jacobian with structural properties that expedite the process of forming the matrix exponential - this suggests that exponential type integrators may be a competitive alternative to the IMEX-RK methods.

While many ETD-RK methods have been developed and studied extensively ([4],[5]), in this paper we design ETD-RK methods specifically for timestepping in HOMME-NH. That is, we develop second and third order methods that minimize computational costs and storage of stage values, while maximizing the stability region relevant to HOMME-NH.

The rest of the paper is structured as follows: Section 2 gives the background of HOMME-NH and ETD-RK methods, and Section 3 investigates how the structure of the Jacobian from HOMME-NH can be exploited to efficiently compute the matrix exponential. Section 4 explores various ETD-RK methods, and Section 5 discusses their performance and implementation in HOMME-NH.

2. Background.

2.1. HOMME-NH.

We begin by presenting a brief summary of the nonhydrostatic atmosphere model HOMME-NH; for a more thorough description, see [8]. HOMME-NH is an energy-conserving atmosphere model, given by the following system of equations:

$$\begin{aligned}
&\frac{\partial \mathbf{u}}{\partial t} + (\nabla_\eta \times \mathbf{u} + 2\Omega) \times \mathbf{u} + \frac{1}{2}\nabla_\eta (\mathbf{u}\cdot\mathbf{u}) + \frac{d\eta}{dt}\frac{\partial \mathbf{u}}{\partial \eta} + \frac{1}{\rho}\nabla_\eta p = 0,\\
&\frac{\partial w}{\partial t} + \mathbf{u}\cdot\nabla_\eta w + \frac{d\eta}{dt}\frac{\partial w}{\partial \eta} + \mathfrak{g}(1-\mu) = 0,\\
&\frac{\partial \phi}{\partial t} + \mathbf{u}\cdot\nabla_\eta \phi + \frac{d\eta}{dt}\frac{\partial \phi}{\partial \eta} - \mathfrak{g} w = 0,
\end{aligned}$$

*Department of Mathematics, University of Kansas, ckrause@ku.edu
†Sandia National Laboratories, asteyer@sandia.gov

$$\frac{\partial \Theta}{\partial t} + \nabla_\eta \cdot (\Theta \mathbf{u}) + \frac{\partial}{\partial \eta}\left(\Theta \frac{d\eta}{dt}\right) = 0,$$
$$\frac{\partial}{\partial t}\left(\frac{\partial \pi}{\partial \eta}\right) + \nabla_\eta \cdot \left(\frac{\partial \pi}{\partial \eta}\mathbf{u}\right) + \frac{\partial}{\partial \eta}\left(\pi \frac{d\eta}{dt}\right) = 0.$$

The equation of state is $\frac{\partial \phi}{\partial \eta} = -R\Theta\Pi/p$. The variables of HOMME-NH are summarized in Table 2.1.

Table 2.1: Variables in HOMME-NH

Variable	Description
$\mathfrak{g}$	Gravitational constant
$\boldsymbol{u} = (u, v)^T$	Horizontal velocity
w	Vertical velocity
ϕ	Geopotential
θ	Potential temperature
p	Nonhydrostatic pressure
π	Hydrostatic pressure
Π	Exner pressure
η	Mass-based hybrid terrain-following vertical coordinate
R	Gas constant
μ	$\frac{\partial p}{\partial \eta} / \frac{\partial \pi}{\partial \eta}$
Θ	$\frac{\partial \pi}{\partial \eta}\theta$

We integrate these equations on a cubed sphere discretization. As discussed in [7], there is a discrepancy in the vertical scale ($\mathcal{O}(10)$ m) and the horizontal scale ($\mathcal{O}(1)$ km), which results in the propagation of numerically stiff acoustic waves in the vertical direction. Consequently, the step-size of standard explicit integrators becomes restricted for stability reasons making them overly expensive. On the other hand, the solves required by standard fully implicit integrators are also prohibitively expensive. In [7], they recognize that the stiff terms of HOMME-NH are only those which involve the coupling of the vertical velocity w, the geopotential ϕ, and μ. This means that the evolution of the state variables can be additively partitioned into the stiff terms s and nonstiff terms n as

$$\omega_t = \begin{bmatrix} \boldsymbol{u}_t \\ w_t \\ \phi_t \\ \theta_t \\ \frac{\partial \pi}{\partial \eta} \end{bmatrix} = g(\omega) \coloneqq s(\omega) + n(\omega), \tag{2.1}$$

where

$$s(\omega) = \begin{bmatrix} 0 \\ -\mathfrak{g}(1-\mu) \\ \mathfrak{g}\mu \\ 0 \\ 0 \end{bmatrix}.$$

The traditional approach in atmospheric modeling is to integrate such a partitioned system with an IMEX-RK time integrator, where the stiff terms are treated with an implicit RK method and the nonstiff terms are treated with an explicit RK method, as described in [7] and [10]. Since the stiff terms involve only the vertically propagating acoustic waves, this is often referred to as a horizontally-explicit, vertically-implicit (HEVI) partitioning.

There are many other ways to integrate a stiff IVP; one such approach is ETD-RK methods. These require an additive partitioning that is slightly different than (2.1). At each time step t_m, we linearize around the state ω_m, which gives the sparse, time-dependent Jacobian

$$L_m = \begin{bmatrix} 0 & & & & \\ & 0 & \mathfrak{g}\frac{\partial \mu}{\partial \phi} & & \\ & \mathfrak{g}I & 0 & & \\ & & & 0 & \\ & & & & 0 \end{bmatrix},$$

where $\dfrac{\partial \mu}{\partial \phi}$ is tridiagonal. As we will see in Section 3, the structure of this Jacobian lends itself to efficient computation of the matrix exponential, making exponential-type integrators an attractive candidate for timestepping in HOMME-NH.

The new additive splitting we use for ETD-RK methods is given by

$$\omega_t = L_m \omega(t_m) + N(\omega(t_m)), \tag{2.2}$$

where $N(\omega(t_m)) = g(\omega(t_m)) - L_m \omega(t_m)$ consists of the nonstiff terms, along with the nonlinear terms of the stiff part. By using this splitting, we are implicitly assuming that most of the stiffness is encompassed in the linear part of the stiff terms.

2.2. Exponential Time Integrators. The construction and analysis of ETD-RK methods have been discussed in depth; see, for example, [4], [1] and [5]. We briefly summarize the theory of ETD-RK methods here.

An s-stage ETD-RK method gives the step update as

$$\omega_{m+1} = e^{L_m \Delta t}\omega_m + \Delta t \sum_{i=1}^{s} b_i N(U_{mi})$$

$$U_{mi} = e^{c_i L_m \Delta t}\omega_m + \Delta t \sum_{j=1}^{i-1} a_{ij} N(U_{mj}),$$

where the $\{a_{ij}\}$, $\{b_i\}$, and $\{c_i\}$ are presented in the form of a Butcher tableau $\begin{array}{c|c} c & A \\ \hline & b^T \end{array}$

Unlike regular Runge Kutta (RK) methods whose Butcher tableaux are filled with constant values, ETD-RK methods use exponential or exponential-type functions of $\Delta t L_m$ for the $\{a_{ij}\}$ and $\{b_i\}$ values. These exponential-type functions are called φ-functions, and they are defined as

$$\varphi_k(z) = \sum_{n=0}^{\infty} \frac{z^n}{(k+n)!}.$$

This series definition of the φ-functions is particularly helpful for analysis purposes. For example, we may consider the underlying RK method, which is found by taking the constant

values of the series expansion of each of the terms in the Butcher tableau of a given ETD-RK method. In implementation, however, it may be more practical to use the recursive definition

$$\varphi_{k+1}(z) = z^{-1}(\varphi_k(z) - \varphi_k(0)), \text{ with } \varphi_0(z) = e^z.$$

Two important characteristics of any numerical timestepper are its order and its stability region, both of which determine the accuracy and size of timestep we are able to use.

DEFINITION 2.1. *A numerical method is said to be* consistent *if the method limits to the true solution as we refine the step-size Δt. Given an s-stage ETD-RK method with Butcher tableau* $\begin{array}{c|c} c & A \\ \hline & b^T \end{array}$*, the conditions required for consistency are*

$$\sum_{i=1}^{s} b_i = \varphi_1, \quad \textit{and}$$
$$\sum_{j=1}^{i-1} a_{ij} = c_i \varphi_1(c_i L \Delta t), \quad i = 1, \ldots, s.$$

DEFINITION 2.2. *A numerical method is said to be* of order p *if the numerical solution ω_{m+1} is such that*

$$\omega(t_{m+1}) - \omega_{m+1} = \mathcal{O}(\Delta t^{p+1}),$$

where $\omega(t_{m+1})$ is the true solution.

Rather than analyze the Taylor expansion of every ETD-RK method, we can use rooted tree analysis to show order conditions for general ETD-RK methods ([4], [1], and [5]). Here we briefly list the order conditions for second and third order methods, referring to previous works for the proof.

THEOREM 2.3. *An s-stage ETD-RK method given by the Butcher tableau* $\begin{array}{c|c} c & A \\ \hline & b^T \end{array}$ *achieves second order if it is consistent and*

$$\sum_{i=2}^{s} c_i b_i = \varphi_2.$$

If the above condition is satisfied, and we also have

$$\sum_{i=2}^{s} c_i^2 b_i = 2\varphi_3, \quad \textit{and} \quad \sum_{i=3}^{s} b_i \sum_{j=2}^{i-1} c_j a_{ij}^{(0)} = \varphi_3,$$

where $a_{ij}^{(0)}$ refers to the constant term in the series expansion of a_{ij}, then the method is third order.

In Section 4, we will present various second and third order methods, and in Section 5 we will discuss their stability and implementation.

3. The Matrix Exponential for HOMME-NH. Clearly, ETD-RK methods rely heavily on the matrix exponential, so it is imperative that we have an efficient and accurate approximation for $e^{\alpha L_m}$, for a given constant α. There are several ways to compute an

approximation to the matrix exponential [6]. We choose to employ a $(2,2)$-Padé approximation with scaling and squaring. The (p,q)-Padé approximation for the matrix exponential is defined as

$$e^X \approx Q_{p,q}^{-1}(X) P_{p,q}(X),$$

where

$$P_{p,q}(X) = \sum_{j=0}^{p} \frac{(p+q-j)!p!}{(p+q)!j!(p-j)!} X^j$$
$$Q_{p,q}(X) = \sum_{j=0}^{q} \frac{(p+q-j)!q!}{(p+q)!j!(q-j)!} (-X)^j.$$

The particular structure of the linearization of the stiff terms in HOMME-NH allows us to form $Q_{p,q}^{-1}(\alpha L_m) P_{p,q}(\alpha L_m)$ efficiently. We follow the same process as used in [2]. That is, for a $(2,2)$-Padé approximation, we can write the matrix exponential as

$$\left(Q_{2,2}(\alpha L_m)\right)^{-1} P_{2,2}(\alpha L_m) \approx e^{\alpha L_m}$$
$$P_{2,2}(\alpha L_m) \approx Q_{2,2}(\alpha L_m) e^{\alpha L_m}.$$

Since we are using a $(2,2)$-Padé approximation, we can factor

$$Q_{2,2}(\alpha L_m) = \kappa \left(\sigma_1 I - \alpha L_m\right)\left(\sigma_2 I - \alpha L_m\right),$$

with constants $\kappa, \sigma_1, \sigma_2$. While the constants σ_1 and σ_2 are independent of αL_m and can be computed explicitly ahead of time, we leave them as variables here, noting that if a different (p,q)-Padé approximation were chosen, the same argument could be used with different constants. We consider just the first factor $(\sigma_1 I - \alpha L_m)$, and denote $R \coloneqq \kappa \left(\sigma_2 I - \alpha L_m\right) e^{\alpha L_m}$. Our equation then becomes

$$P_{2,2}(\alpha L_m) = \begin{pmatrix} \sigma_1 I & -\alpha \mathfrak{g} \frac{\partial \mu}{\partial \phi} \\ -\alpha \mathfrak{g} I & \sigma_1 I \end{pmatrix} R.$$

Left multiplying by a permutation matrix gives us

$$\tilde{P}_{2,2}(\alpha L_m) = \begin{pmatrix} \sigma_1 I & -\alpha \mathfrak{g} \frac{\partial \mu}{\partial \phi} \\ 0 & \sigma_1 I - (\alpha \mathfrak{g})^2 \sigma_1^{-1} \frac{\partial \mu}{\partial \phi} \end{pmatrix} \begin{pmatrix} R_1 \\ R_2 \end{pmatrix}.$$

This formulation allows us to solve for R_2 with just a tridiagonal solve, and R_1 with back substitution. On the other hand, we also have

$$R = \kappa \left(\sigma_2 I - \alpha L_m\right) e^{\alpha L_m}.$$

Repeating these steps allows us to solve for $e^{\alpha L_m}$ with just two tridiagonal solves, which is much cheaper than finding $Q_{2,2}(\alpha L_m)$ explicitly.

The Padé approximation is most accurate when the complex eigenvalues of the operator lie within the unit disk. We use the scaling and squaring technique [3] to rescale the spectrum of the operator to the unit disk. Using the matrix relation $\exp(X) = \left(\exp(X/2^k)\right)^{2^k}$, we scale αL_m by a power of 2 before computing the matrix exponential. This result is then repeatedly squared until we get the desired $e^{\alpha L_m}$.

4. Designing ETD-RK Methods for HOMME-NH. When selecting an ETD-RK method for implementation, two things we consider are efficient computation of stage values and minimal storage. We do not claim that the following methods are unique to this paper, but their derivation was original work with these goals in mind.

Here and throughout, we use the naming convention ETD-RK-i-j, where i gives the order of the method and j is the number of stages. In the case where we consider more than one method with a given order and number of stages, we denote the different methods by a subsequent letter, $a, b, \ldots$. The first method we consider is a second order ETD-RK method, which we call ETD-RK-2-3a. It is given by the following Butcher tableau:

$$\begin{array}{c|ccc} 0 & & & \\ \frac{1}{2} & \frac{1}{2}\varphi_1^2 & & \\ 1 & 0 & \varphi_1^3 & \\ \hline & \varphi_1 & -2\varphi_2 & 2\varphi_2 \end{array} \tag{4.1}$$

where $\varphi_i^j = \varphi_i(c_j L_m \Delta t)$, and $\varphi_i = \varphi_i(L_m \Delta t)$. One downfall to (4.1) is that it requires the storage of all of the stage values in order to compute the final state update. We can instead consider another second order method, ETD-RK-2-3b, given by

$$\begin{array}{c|ccc} 0 & & & \\ \frac{1}{2} & \frac{1}{2}\varphi_1^2 & & \\ 1 & 0 & \varphi_1^3 & \\ \hline & \varphi_1 - \varphi_2 & 0 & \varphi_2 \end{array} \tag{4.2}$$

More computation is required in order to achieve higher order. In fact, a third order method with entries only on the main subdiagonal will require at least 4 stages. One such method is given by ETD-RK-3-4a:

$$\begin{array}{c|cccc} 0 & & & & \\ \frac{1}{2} & \frac{1}{2}\varphi_1^2 & & & \\ \frac{1}{2} & & \frac{1}{2}\varphi_1^3 & & \\ 1 & & & \varphi_1^4 & \\ \hline & \varphi_1 - 3\varphi_2 + 4\varphi_3 & 2\varphi_2 - 4\varphi_3 & 2\varphi_2 - 4\varphi_3 & -\varphi_2 + 4\varphi_3 \end{array} \tag{4.3}$$

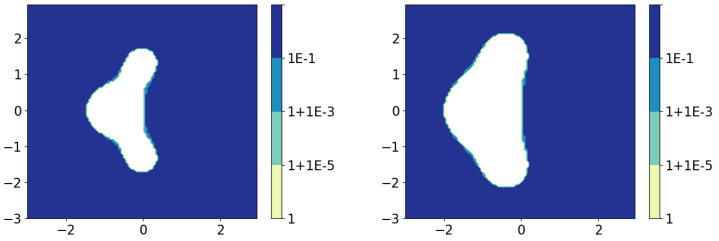
If we change the constants in (4.3), we can derive another third order method with the same structure:

$$\begin{array}{c|cccc} 0 & & & & \\ \frac{1}{4} & \frac{1}{4}\varphi_1^2 & & & \\ \frac{3}{4} & & \frac{3}{4}\varphi_1^3 & & \\ 1 & & & \varphi_1^4 & \\ \hline & \varphi_1 - \frac{17}{3}\varphi_2 + \frac{32}{3}\varphi_3 & \frac{20}{3}\varphi_2 - 16\varphi_3 & \frac{-4}{3}\varphi_2 + \frac{16}{3}\varphi_3 & \frac{1}{3}\varphi_2 \end{array} \tag{4.4}$$

By checking the conditions of Theorem 2.3, we can show that each of these methods achieves the desired order.

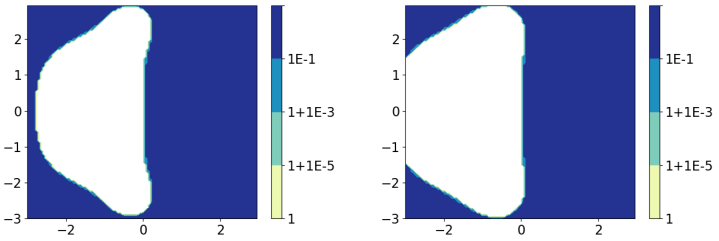
5. Results. In addition to considering the computational cost and storage requirements of various ETD-RK methods, we also are interested in the stability regions of given methods. Because the stiff and nonstiff terms of HOMME-NH are hyperbolic, the linearization has purely imaginary eigenvalues, so we wish to use a method that has maximum stability on the imaginary axis. While the stability analysis of the coupling of the underlying RK method with the exponential part is beyond the scope of this paper, we recognize that a

necessary condition for stability of an IMEX-RK method to integrate HOMME-NH is that the underlying method has some stability on the imaginary axis. The stability regions of the underlying RK methods for (4.1) and (4.2) are shown in white in Figure 5.1. A similar comparison of the third order methods is shown in Figure 5.2.



(a) ETD-RK-2-3a (b) ETD-RK-2-3b

Fig. 5.1: Stability plots for second order ETD-RK methods. The white region denotes the stable area of the underlying RK method. We wish to use a method with maximal stability on the imaginary axis for HOMME-NH.



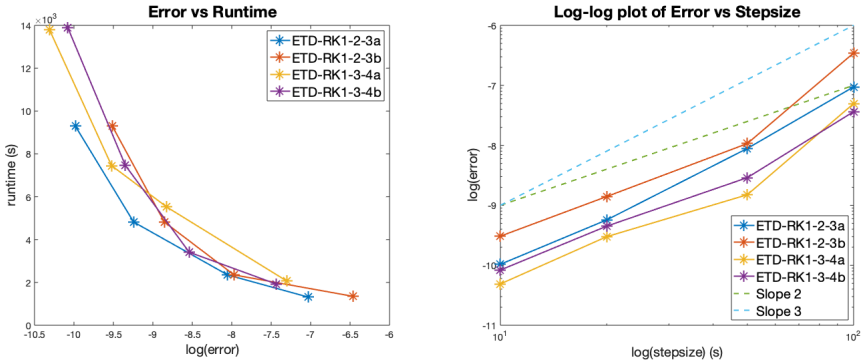
(a) ETD-RK-3-4a (b) ETD-RK-3-4b

Fig. 5.2: Stability plots for third order ETD-RK methods. The stability regions of the underlying RK methods for these four stage ETD-RK methods are considerably larger than in Figure 5.1.

It is worth noting that even though ETD-RK-2-3a and ETD-RK-2-3b are both second order methods with three stages, their stability regions look quite different: ETD-RK-2-3b has a much larger stability region than ETD-RK-2-3a. It is also well known that increasing the number of stages often also increases the stability region, and we see that phenomenon occurring when we compare the stability regions of ETD-RK-2-3a and ETD-RK-2-3b with those of ETD-RK-3-4a and ETD-RK-3-4b. The trade-off is, of course, that the methods with four stages require greater computation time.

To truly compare the utility of these methods, we must consider their overall efficiency and accuracy. One way to do this is to look at a convergence test (step-size vs relative error) along with an efficiency study of error vs runtime. This is shown in Figure 5.3. We run our time integrators with the 2012 Dynamical Core Model Intercomparison Project (DCMIP)

test case 3.1, which is a potential temperature perturbation along the equator of a small planet, resulting in gravity waves [9]. The reference solution is a second order implicit, third order explicit IMEX-RK method with a step size of 0.01 s.



(a) Efficiency Study (b) Convergence Test

Fig. 5.3: Results of a 1000 second integration of test case DCMIP 2012-3.1 on a small planet. Lines with slopes of 2 and 3 are included in the convergence test for comparison.

6. Conclusions. The linearization of the stiff terms of HOMME-NH yielded a sparse Jacobian with a block tridiagonal structure. This allowed for efficient computation of the matrix exponential, making ETD-RK methods a viable time integrator for HOMME-NH. In order for ETD-RK methods to be competitive with the current IMEX-RK methods, the ETD-RK methods needed to store minimal stage values, and the computation of each stage value needed to be as efficient as possible. In this paper, we presented several methods that met these criteria. Their efficiency and accuracy was compared on a 1000 second run of test case DC12.3.1. All of these methods performed reasonably well, and the stability regions of the underlying RK methods were satisfactory.

We draw special attention to ETD-RK2-3a, which seems to have some of the best accuracy and fastest computation time. If accuracy is prioritized above all else, then the recommended method to use would be ETD-RK-3a with a small step-size.

It remains to be seen how these methods compare directly with the IMEX-RK methods. One drawback to the ETD-RK methods is that the scaling and squaring for the matrix exponential required many matrix multiplications. If another approximation to the matrix exponential is more efficient than the $(2,2)$-Padé approximation with scaling and squaring, the ETD-RK methods might be a competitive alternative to the IMEX-RK time integrators in HOMME-NH.

REFERENCES

[1] H. Berland, B. Owren, and B. Skaflestad, *B-series and order conditions for exponential integrators*, SIAM Journal on Numerical Analysis, 43 (2005), pp. 1715–1727.

[2] C. Krause and A. Steyer, *Exponential integrators for the homme-nh nonhydrostatic atmosphere model*, Center for Computing Research Summer Proceedings 2019, (2019).

[3] N. Higham, *The scaling and squaring method for the matrix exponential revisited*, SIAM Journal on Matrix Analysis and Applications, 26 (2005), pp. 1179–1193.

[4] Hochbruck, M. and Ostermann, A., *Explicit exponential Runge-Kutta methods for semilinear parabolic problems*, SIAM J. Numer. Anal., 43 (2005), pp. 1069–1090.

[5] S. Koikari, *Rooted tree analysis of runge–kutta methods with exact treatment of linear terms*, Journal of Computational and Applied Mathematics, 177 (2005), pp. 427 – 453.

[6] C. Moler and C. Van Loan, *Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later*, SIAM Review, 45 (2003), pp. 3–49.

[7] A. Steyer, C. J. Vogl, M. Taylor, and O. Guba, *Efficient imex runge-kutta methods for nonhydrostatic dynamics*, arXiv preprint arXiv:1906.07219, (2019).

[8] Taylor, M., Guba, O., Steyer, A., Ullrich, P., Hall, D., and C. Eldred, *An energy consistent discretization of the nonhydrostatic equations in primitive variables*, Preprint, (2019).

[9] P. A. Ullrich, C. Jablonowski, J. Kent, P. H. Lauritzen, R. D. Nair, and M. A. Taylor, *Dynamical core model intercomparison project (dcmip) test case document*, (2012).

[10] C. J. Vogl, A. Steyer, D. R. Reynolds, P. A. Ullrich, and C. S. Woodward, *Evaluation of implicit-explicit additive runge-kutta integrators for the homme-nh dynamical core*, arXiv preprint arXiv:1904.10115, (2019).

LIQUID HYDROGEN STORAGE TANK PRESSURE RELIEF SIMULATIONS USING NETWORK FLOW MODELING

DEREK M. MACHALEK*, GABRIELA BRAN-ANLEU†, AND ETHAN S. HECHT‡

Abstract. For hydrogen vehicles to gain further acceptance, hydrogen fueling stations must be wide-ranging and safe. At larger fueling stations, hydrogen is often stored and delivered in the liquid phase because of its high energy density and fast transfer. In a liquid hydrogen storage tank, hydrogen vapor exists above the cryogenic liquid. A common modeling assumption of a liquid hydrogen tank is thermodynamic equilibrium. However, in hazardous scenarios this assumption may not hold. A non-equilibrium storage tank model that included different boiling regimes was developed and used to examine 5 different scenarios. The case of normal boiloff was used to validate the model. Scenarios with high conductivity through the insulation layer (e.g., if vacuum were lost in the insulation layer), and high outside temperatures (e.g., if there were a fire impinging on the tank), and combinations of these scenarios were used to exercise the model. The simulations showed that there are differences between an equilibrium and a non-equilibrium model as a tank depressurizes through a burst disk, boiling can be important in modeling non-equilibrium hydrogen tanks, and that the combination of a pressure relief valve and burst disk are sufficient to depressurize a liquid hydrogen tank under these abnormal scenarios. These and additional scenarios are of interest to the safety codes and standards community, and can be used to ensure hydrogen's safe use at hydrogen fueling stations.

1. Introduction. Hydrogen has significant potential to impact the transportation industry, one of the largest consumers of fossil fuel. Hydrogen fuel cell vehicles can provide a clean way to travel long distances. In order to gain adoption, hydrogen refueling stations must be ubiquitous and safe. Currently, aspects of safety codes and standards for liquid hydrogen refueling stations such as separation distances lack a well documented basis. The separation distances are critical to hydrogen energy adoption, because the footprint of the refueling station must be large enough to be safe, but small enough that more plots of land are viable. The objective of this research is to enable simulations of hazardous scenarios that hydrogen refueling stations may encounter. Specifically, a model for a liquid hydrogen storage tank is needed, so that the rate of venting and blowdown of the tank can be accurately predicted for a variety of circumstances, including normal conditions and off-normal conditions such as a failure of the vacuum insulation.

Figure 1.1 shows a liquid hydrogen storage tank diagram. The tank contains both liquid hydrogen and vapor hydrogen. In the model, these layers are separated by a thin massless film of saturated vapor. Additional components involved in the hydrogen release scenarios of interest in this study are shown in Figure 1.1, specifically the burst disc and pressure relief valve. The rest of the piping and valves for loading the tank with hydrogen or discharging the hydrogen to vehicles are not shown. The wall of the liquid hydrogen storage tank is composed of an interior steel shell, a vacuum space with multi-layer insulation (MLI) material, and an external steel shell. The vacuum and low thermal conductivity of the MLI significantly reduce the heat transfer from the outside environment. As the liquid storage tank sits under the sun for days, the temperature of the liquid hydrogen will increase, causing some liquid hydrogen to evaporate and increase the pressure of the tank. For safety reasons, the tank is designed to have normal boil-off flow through the relief valve to ensure the pressure inside of the tank does not increase beyond the working pressure. In these simulations, the pressure relief valve will open at 3.1 bar and reseal when the tank pressure drops to 2.9 bar. If the pressure relief valve malfunctions or is not able to reduce the pressure, the burst disc acts as a secondary safety mechanism to prevent the tank from exceeding the maximum allowable

*University of Utah, derek.machalek@utah.edu
†Sandia National Laboratories, gabrana@sandia.gov
‡Sandia National Laboratories, ehecht@sandia.gov

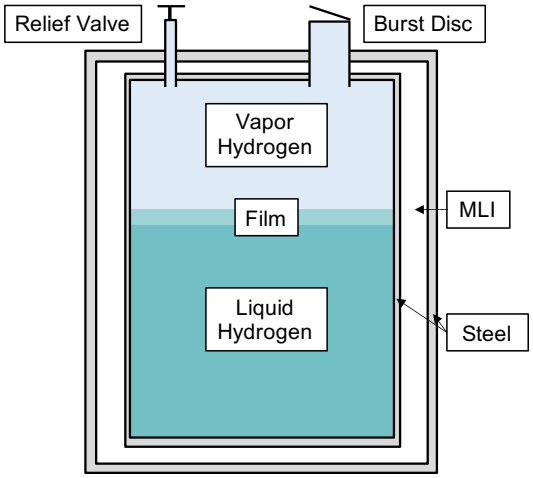


Fig. 1.1: Diagram of liquid hydrogen storage tank.

working pressure. In these simulations, the burst disc opens at a hydrogen vapor pressure of 4 bar, and has a much larger opening than the pressure relief valve. Once the burst disc opens, it does not close again.

Cryogenic tank dynamics have been previously researched for rocket fueling applications [5, 7, 8]. Estey et al. [5] developed a thermodynamic non-equilibrium model to characterize the blowdown process of a propellant tank, where they solve for the mass and energy equations of the liquid and vapor phase. They assumed a massless liquid-vapor interface where heat transfer between the liquid and vapor happens only by convection. The authors in [7, 8] developed a model for the filling of a cryogenic tank and showed that heat transfer by conduction is also important when dynamic condensation blocking is present. Dynamic condensation blocking happens when when heat conduction from the liquid-vapor interface to the liquid is not fast enough that the temperature of the liquid-vapor interface increases until condensation is stopped by evaporation. Petitpas [9] modified the Matlab model developed by Osipov et al. [8] to model the boil-off losses during transfer of liquid hydrogen from a trailer to a storage tank located at a hydrogen refueling station. We build off of these previous works and present a model that also includes heat transfer via boiling, which becomes important with high heat flow into the tank. We also include a more detailed wall heat transfer model than the previous works.

In this study, four liquid hydrogen release scenarios under abnormal conditions were investigated to demonstrate the model capabilities: 1) vacuum loss in the MLI layer, 2) an external fire engulfing the storage tank, 3) loss of vacuum and an engulfing fire, and 4) high heat conduction through the insulation layer. The key metrics explored for the hydrogen releases are how fast hydrogen is released and whether or not the pressure relief valve is sufficient to lower the pressure of the tank to the rated tank pressure. Both thermodynamic equilibrium and non-equilibrium models of the tank were explored to determine in what scenarios the equilibrium assumption was valid and when it was not.

2. Model. The physics models were implemented in MassTran, a Sandia developed python software [2]. MassTran uses the CoolProp [1] package in Python to maintain the Helmholtz Equation of State (EOS). In the equilibrium model, the liquid and gas are assumed to be in thermodynamic equilibrium (i.e., at the same temperature). That constraint

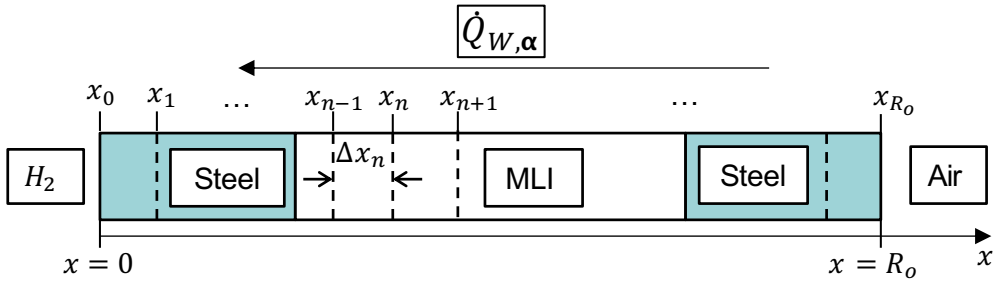


Fig. 2.1: 1-D wall heat transfer model.

allows the vapor and liquid to be treated as a single control volume with only one continuity equation. In contrast, in the non-equilibrium case the liquid and vapor will not be at the same temperature, so the liquid and vapor mass balances must be treated separately. A detailed wall heat transfer model, which includes wall boiling for high heat transfer scenarios, was also included in the model.

2.1. Wall Heat Transfer Model. The 1-D wall heat transfer is shown in Figure 2.1. The dashed vertical lines represent the finite volumes used to solve the problem numerically. The 1-D energy equation to calculate the wall temperature, T_w, along the radial coordinates, x, is shown in (2.1).

$$\rho_w C_{p,w} \frac{\partial T_w}{\partial t} = \kappa_w \frac{\partial^2 T_w}{\partial x^2} \tag{2.1}$$

where ρ_w is the density of the wall, $C_{p,w}$ is the specific heat of the wall, and κ_w is thermal conductivity of the wall. Boundary conditions are shown in (2.2) and (2.3) are imposed at the hydrogen-wall interface and wall-air interface, respectively.

$$\left.\frac{\partial T_w}{\partial t}\right|_{x=0} = \frac{h_i}{\rho_w C_{p,w}} (T_w(x=0) - T_\alpha) \tag{2.2}$$

$$\left.\frac{\partial T_w}{\partial t}\right|_{x=R_0} = \frac{h_{air}}{\rho_w C_{p,w}} (T_w(x=R_0) - T_{air}) \tag{2.3}$$

where h_i is the convective heat transfer coefficient between the hydrogen inside the tank and the inner wall, and h_{air} is the convective heat transfer coefficient between air and the outer wall. The wall density is ρ_w, $C_{p,w}$ is the specific heat of the wall, T_{air} is the temperature of air, and T_α is the temperature of phase α. R_0 is the outer radius of the outer steel layer. Details about developing implicit energy balances for the wall are shown in Appendix A.3.

Heat transfer mechanisms between the inner wall of the storage tank and the hydrogen are shown in red in Figure 2.2. The storage tank was assumed to be a vertical cylinder. The heat transfer between the wall and the hydrogen depends on the phase of the hydrogen and the surface geometry. Separate wall heat transfer was calculated for the following mechanisms: 1) the top surface of the wall and H_2 vapor, $\dot{Q}_{T,V}$, 2) the side wall and H_2 vapor $\dot{Q}_{S,V}$, 3) the tank side surface and liquid H_2, $\dot{Q}_{S,L}$, and 4) the tank bottom surface and liquid H_2 $\dot{Q}_{B,L}$. The heat transfer rates were summed to calculate overall wall heat transfer to the liquid and vapor ($\dot{Q}_{L,W} = \dot{Q}_{S,V} + \dot{Q}_{S,V}$ and $\dot{Q}_{L,W} = \dot{Q}_{S,L} + \dot{Q}_{B,L}$).

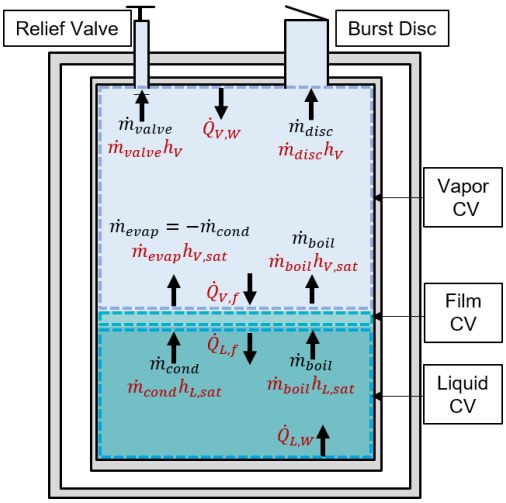


Fig. 2.2: Mass and energy flown in liquid and vapor control volumes (CVs) in a liquid hydrogen tank during hydrogen releases through a valve or burst disk.

The heat transfer between the wall and the liquid hydrogen, $\dot{Q}_{L,W}$, is shown in (2.4), and the vapor phase heat transfer, $\dot{Q}_{V,W}$, is calculated with equation (2.5).

$$\dot{Q}_{L,W} = \frac{\kappa_L}{fH}\mathrm{Nu}_{S,L}(fH\pi D_h)(T_w - T_L) + \frac{\kappa_L}{D_h/2}\mathrm{Nu}_{B,L}\left(\frac{\pi D_h^2}{4}\right)(T_w - T_L) \tag{2.4}$$

$$\dot{Q}_{V,W} = \frac{\kappa_v}{(1-f)H}\mathrm{Nu}_{S,V}((1-f)H\pi D_h)(T_w - T_V) + \frac{\kappa_V}{D_h/2}\mathrm{Nu}_{T,V}\left(\frac{\pi D_h^2}{4}\right)(T_w - T_V) \tag{2.5}$$

The heat transfer correlations used in this model are listed in (2.6)-(2.9) [6].

$$\mathrm{Nu}_{S,L} = \left(0.825 + \frac{0.387\mathrm{Ra}^{1/6}}{(1 + (0.492/\mathrm{Pr})^{9/16)})^{8/27}}\right)^2 \tag{2.6}$$

$$\mathrm{Nu}_{B,L} = 0.\mathrm{Ra}^{1/3} \tag{2.7}$$

$$\mathrm{Nu}_{S,V} = \left(0.825 + \frac{0.387\mathrm{Ra}^{1/6}}{(1 + (0.492/\mathrm{Pr})^{9/16)})^{8/27}}\right)^2 \tag{2.8}$$

$$\mathrm{Nu}_{T,V} = 0.52\mathrm{Ra}^{1/5} \tag{2.9}$$

In these equations, Ra is the Rayleigh number, Pr is the Prandtl number, D_h is the tank diameter (and characteristic length in the Rayleigh numbers for Eqs. 2.7 and 2.9), f is the fraction fill of liquid, H is the overall tank height (and the height of the liquid or gas is the characteristic length in the Rayleigh numbers for Eqs. 2.6 and 2.8), and T_L and T_V are the

liquid and vapor hydrogen temperatures. The correlations for external flow over a flat plate, (2.7) and (2.9), are valid for Rayleigh numbers less than 10^9 and 10^{11}, respectively. The external flow over a vertical plate correlations, (2.6) and (2.8) are for the entire Rayleigh range [3]. It was assumed that the surface areas of the tank are reasonably large that these external flow correlations can be used for this study. The heat transfer from the wall to the hydrogen is calculated from these correlations.

2.2. Governing Equations. In the non-equilibrium model, a set of governing equations is defined for the liquid phase control volume (CV), and a different set is defined for the vapor phase CV as shown in Figure 2.2. The liquid mass balance is captured in (2.10),

$$\frac{dm_L}{dt} = \dot{m}_{cond} - \dot{m}_{boil} \tag{2.10}$$

where m_L is the mass of the liquid. The condensation rate, $\dot{m}_{cond}$, is described in Section 2.3, and the boiling rate, $\dot{m}_{boil}$, is described in Section 2.4. The vapor mass balance is captured in (2.11),

$$\frac{dm_V}{dt} = \dot{m}_{evap} + \dot{m}_{boil} - \dot{m}_{valve} - \dot{m}_{disc} \tag{2.11}$$

where m_V is the mass of the gas. The evaporation rate, $\dot{m}_{evap}$, is described in Section 2.3, and the mass flow rates leaving the tank, $\dot{m}_{valve}$ and $\dot{m}_{disc}$, are described in Section 2.5.

The liquid energy balance is captured in (2.12).

$$\frac{dU_L}{dt} = \dot{Q}_{L,f} + \dot{Q}_{L,W} + \dot{m}_{cond}h_{L,sat} - \dot{m}_{boil}h_{L,sat} \tag{2.12}$$

where U_L is the internal energy of the liquid, and $\dot{Q}_{L,f}$ is the heat transfer rate from the film to the liquid. Similarly, the vapor energy balance is captured in (2.13).

$$\frac{dU_V}{dt} = \dot{Q}_{V,f} + \dot{Q}_{V,W} + \dot{m}_{boil}h_{V,sat} + \dot{m}_{evap}h_{V,sat} - (\dot{m}_{valve} + \dot{m}_{disc})h_V \tag{2.13}$$

In these equations, $\dot{Q}_{V,f}$ is the heat transfer rate from the vapor to the film, and h is the enthalpy of the liquid (L) or vapor (V) phase at the actual or saturated (sat) conditions.

The equations for the equilibrium tank are simplified into a single mass and energy balance. Further, the mass and energy transfer at the interface of the two phases is computed with the thermodynamic equilibrium constraint. Therefore, the $\dot{m}_{ec}$, $\dot{m}_{boil}$, and $\dot{m}_{cond}$ terms are neglected. The equilibrium mass balance and energy balances are found by ignoring the interface terms and adding the two non-equilibrium mass balances together and the two non-equilibrium energy balances together. The mass and energy balances for the equilibrium model are shown in (2.14) and (2.15), respectively.

$$\frac{dm}{dt} = -\dot{m}_{valve} - \dot{m}_{disc} \tag{2.14}$$

$$\frac{dU}{dt} = \dot{Q}_{L,W} + \dot{Q}_{V,W} - (\dot{m}_{valve} + \dot{m}_{disc})h_V \tag{2.15}$$

2.3. Liquid-vapor Interface Evaporation and Condensation. The liquid and vapor are modeled as being separated by a thin mass-less vapor film at the saturation temperature of the vapor phase (i.e., $T_f = T_v$). The evaporation rate, $\dot{m}_{evap}$, and condensation rate, $\dot{m}_{cond}$, are limited by the heat transfer to the liquid-vapor interface. An energy balance on the thin vapor film can be performed to obtain the evaporation mass flow rate, $\dot{m}_{evap}$ (2.16).

$$\dot{m}_{evap} = -\frac{\dot{Q}_{L,f} + \dot{Q}_{V,f}}{h_{vap}} \tag{2.16}$$

The heat of vaporization, $h_{vap}(T_f) = h_{V,f} - h_{L,f}$, is defined at the film temperature, T_f. The heat transfer between the film and the phase (α = L or V), $\dot{Q}_\alpha$, is via conduction and in some cases via convection. While conductive heat transfer to the thin film is always happening, the only convection mechanism, natural convection only occurs in the vapor when the vapor is colder than the film or in the liquid when the liquid is hotter than the film (i.e., a higher density fluid is on top of a lower density fluid within a given phase). The heat transfer from either phase to the film is the summation of the heat transfer due to conduction, $\dot{Q}_{\alpha,cd}$, and convection, $\dot{Q}_{\alpha,cv}$ ($\dot{Q}_{\alpha,f} = \dot{Q}_{\alpha,cd} + \dot{Q}_{\alpha,cv}$). A Boundary Layer (BL) model was developed by Osipov and Muratov [7], and later modified by Petitpas [9], to account for the temperature gradients in the liquid and gas volumes. However, the simulation was highly sensitive to BL lengths, and the authors do not justify their length selection. As a result, a simpler conduction correlation for the conductive heat transfer between the film and hydrogen in phase α is shown in (2.17) [4].

$$\dot{Q}_{\alpha,cd} = \left(\frac{\kappa_\alpha C_{v,\alpha} \rho_\alpha}{\pi}\right)^{1/2} A_c (T_\alpha - T_f) \tag{2.17}$$

The equation for natural convection from the is demonstrated in equation (2.18).

$$\dot{Q}_{\alpha,cv} = 0.156 \left(\frac{g\beta\rho_\alpha^2 (T_\alpha - T_F)}{\kappa_\alpha \mu_\alpha}\right)^{1/3} A_c (T_\alpha - T_f) \tag{2.18}$$

In these equations, κ is the thermal conductivity, C_v is the heat capacity, ρ is the density, and T_α is the temperature of the phase α (liquid or gas). T_f is the temperature of the film, A_c is the cross sectional area of the film interface, g is the gravitational constant, β is the thermal expansion coefficient, and μ is the dynamic viscosity.

The condensation rate is the same as the evaporation rate, but in the opposite direction.

$$\dot{m}_{cond} = -\dot{m}_{evap} \tag{2.19}$$

The condensation rate might be adjusted depending on the density calculation. The density is first calculated by assuming that both phases are at the same pressure. A solver is used to find the pressure, given the internal energy of both phases as a result of the solution to the energy equations (Eqs. 2.12 and 2.13). With the density and mass (found by solving the mass balance equations, Eqs. 2.10 and 2.11) of each phase known each phase volume can be calculated. The phase volumes must add up to the total tank volume ($V_V + V_L = V_{Tank}$). If the liquid is at the saturation pressure (has a quality between 0 and 1), the condensation rate is adjusted. The pressure is then recalculated where the liquid is saturated and the remaining tank volume is assumed to be filled with the vapor. Note that sometimes it would be preferable to assume that the vapor is saturated. However, the solver runs into issues since changing the pressure does not greatly impact liquid density. Therefore, when the

Table 2.1: Hydrogen boiling regimes.

Regime	ΔT (K)	$\dot{q}_{boil}$ (W/m^2)
Convective	0-0.1	$0.16\mathrm{Ra}^{1/3}\Delta T$
Nucleate boiling	0.1-3	$6309\Delta T^{2.52}$
Critical heat flux	3	$(0.18-0.14(P/P_C)^{5.68})h_{vap}\rho_V\left(\frac{g\sigma(\rho_L-\rho_V)}{\rho_V^2}\right)^{1/4}$
Transition	3-15	$\dot{Q}_{CHF}-\frac{\Delta T-\Delta T_{CHF}}{\Delta T-\Delta T_{MHF}}(\dot{Q}_{CHF}-\dot{Q}_{MHF})$
Minimum heat flux	15	$0.31h_{vap}\rho_V\left(\frac{g\sigma(\rho_L-\rho_V)}{(\rho_L+\rho_V)^2}\right)^{1/4}$
Film boiling	>15	$f(D,\rho,g,\kappa,\Delta T)$

liquid occupies the remaining volume after the vapor phase volume is set, the liquid has drastic changed in volume and as a result, it has unreasonable properties (e.g., very high temperature). One area to explore is handling of subcooled vapor/superheated liquid to meet the density and volume constraint.

After this calculation, the vapor phase may have a quality less than 1. This sometimes occurs when the pressure is recalculated if the liquid phase density (at saturation) is low enough to expand and confine the vapor phase into a small volume. The resultant density of the vapor phase indicates a liquid vapor mixture. In order to get the vapor phase back to saturation, a first order condensation term between the saturation quality of 1, and the actual quality is used to drive the liquid component of the vapor phase back to the actual liquid phase. The additional condensation $\dot{m}_{cond,add}$ is calculated in (2.20).

$$\dot{m}_{cond,add} = (1-\chi_V)m_V \tag{2.20}$$

Where χ_V is the quality of the vapor and m_V is the mass of vapor.

2.4. Boiling. When the liquid becomes saturated and the temperature difference (ΔT) between the wall and the liquid is large enough, pool boiling will occur. This is a secondary form of heat and mass transfer between the liquid and vapor phases. There are four regimes of pool boiling. Table 2.1 shows the temperature difference and heat transfer function for each of those regimes as well as the critical heat flux (CHF) and minimum heat flux (MHF) [12]. Before boiling was added to the model, in high heat transfer scenarios the liquid in the tank became very hot because the liquid absorbed heat from the wall quickly, but the evaporation rate and associated energy loss through the film was low. By adding boiling to the model, the rapid evaporation rate is captured.

The film boiling regime was not implemented in the code because the temperature difference between the wall and the hydrogen is not expected to be larger than 15 K, even in abnormal heat transfer cases. The expression for the film boiling critical heat flux can be found in [12].

The boiling rate, $\dot{m}_{boil}$, is calculated by dividing the heat transfer due to boiling by the heat of vaporization, since the liquid is saturated,

$$\dot{m}_{boil} = \frac{\dot{q}_{boil}A_{s,L}}{h_{vap}} \tag{2.21}$$

where $A_{s,L}$ is the surface area of the liquid to wall interface.

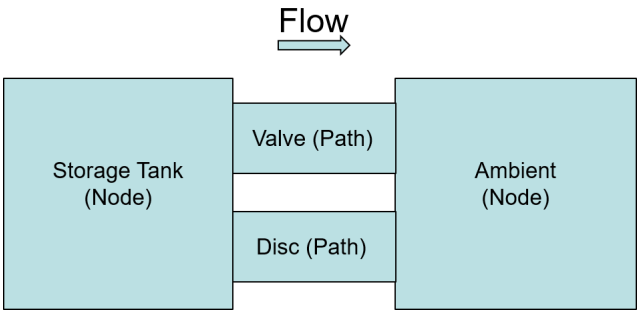


Fig. 2.3: Network flow modeling diagram for hydrogen storage tank.

2.5. Mass Leaving the Tank. Mass leaves the tank through the pressure relief valve $\dot{m}_{valve}$ and the burst disc $\dot{m}_{disc}$ when they are open. The pressure relief valve opens at 3.1 bar and closes at 2.9 bar after hydrogen has been released. The burst disc opens at 4 bar and never closes. They are controlled with state events that track if a pressure triggering event has occurred, which causes them to switch from open to closed or vice-versa.

2.6. Network Flow Modeling - MassTran. This simulation was run under the paradigm of a network flow model. In this work, the framework of MassTran [2] was used. MassTran is able to model compressible flows in networks consisting of pressure vessels, connecting tubing, orifices, valves, and flow branches. Components in MassTran are modeled as nodes or paths. Mass and energy balances are handled in the nodes. Momentum balances are handled in the paths. The diagram of network flow modeling for the hydrogen storage tank is shown in Figure 2.3. The storage tank and environment are modeled as nodes, and the valve and burst disc are modeled as paths. The paths are used to calculate the mass flow rates at the valve and burst disc by solving the momentum equation. The equations for the hydrogen storage tank are developed into nodes for MassTran for the equilibrium and non-equilibrium cases. As previously mentioned, the valve and orifice components already exist in MassTran and were slightly modified to accommodate the pressure based openings for the hydrogen releases through the pressure relief valve. Finally, the ambient node serves as a sink at atmospheric pressure.

3. Validation. The hydrogen storage tank simulations were run with the specifications of a cylindrical hydrogen storage tank (see Figure 1.1) located at Lawrence Livermore National Laboratory. Experimental results from the tank were available for validation of the simulations. Table 3.1 shows the properties of the hydrogen storage tank used in the simulations. Table 3.2 lists the parameters used for the heat transfer tank wall model.

The thermal properties of the wall materials were assumed constant. The thermal properties of the inner steel layer were evaluated at 20K, while the properties of the outer steel layer were evaluated at 300 K. The heat capacity and density of MLI were not known and set to very low values. Effectively, the MLI does not store or discharge any energy, it only serves to resist heat transfer from the two steel shells. The thermal conductivity is estimated to be on the order of 10^{-5} W/m-k [10]. In the electrical resistivity analogy to heat transfer, the heat transfer between steel shells through the MLI is so large that it is effectively the only resistance to heat transfer.

For validation, the tank described by Petitpas [9] with normal boil-off was simulated. As the liquid hydrogen tank sits unused under weather conditions, the liquid hydrogen starts

Table 3.1: Tank Properties.

Component	Value	Units
Tank diameter	2	m
Tank height	3.97	m
Liquid fill	0.8	-
Safety valve open pressure	3.1	bar
Safety valve close pressure	2.9	bar
Safety valve diameter	0.005	m
Burst disc open pressure	4	bar
Burst disc diameter	0.038	m

Table 3.2: Parameters for heat transfer wall model.

Component	Value	Units
Interior steel thickness	0.0111	m
Interior steel heat capacity	25	J/kg-K
Interior steel density	8050	kg/m^3
Interior steel thermal conductivity	3	W/m-K
MLI thickness	0.0508	m
MLI heat capacity	0.1	J/kg-K
MLI density	0.1	kg/m^3
MLI thermal conductivity	20^{-5}	W/m-K
Exterior steel thickness	0.0038	m
Exterior steel heat capacity	450	J/kg-K
Exterior steel density	8050	kg/m^3
Exterior steel thermal conductivity	15	W/m-K
Ambient temperature	300	K
Ambient heat transfer coefficient	10	W/m^2-K

slowly heating up until the vapor pressure inside the tank reaches 3.1 bar. At this point, the pressure relief valves opens to lower the pressure inside of the tank to 2.9 bars. Once the tank reaches a pressure of 2.9 bar, the pressure relief valve closes.

Figure 3.1 shows a simulation for the volume fraction of liquid in the tank for the experimental results and the equilibrium model of the tank. The equilibrium model was used because of computation efficiency and because it shows similar results to the non-equilibrium model (discussed later) for slow tank heat transfer. Since the simulation models the case of freshly loaded hydrogen to the storage tank, there is a temporal difference with the experimental data. The simulated hydrogen first heats up, expands, pressurizes and begins to vent at 3.1 bar of pressure. After the venting of the simulation has reached the same volume fill as the experiments, the experiments are overlaid onto the plot, at approximately day 13.5. The experimental loss of hydrogen from the storage tank is approximately the same as the simulation for the first six days of venting. However, after that point the venting in the experimental tank starts to slow down. The heat transfer to the tank, which drives pressurization and mass transfer out of the tank, appears to be a function of the liquid volume [9].

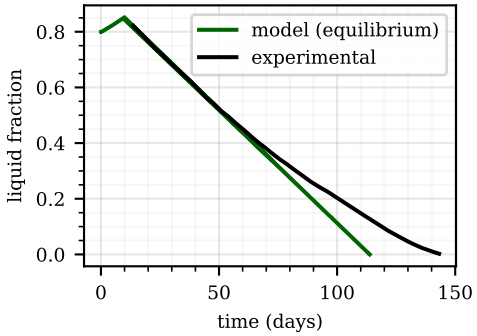
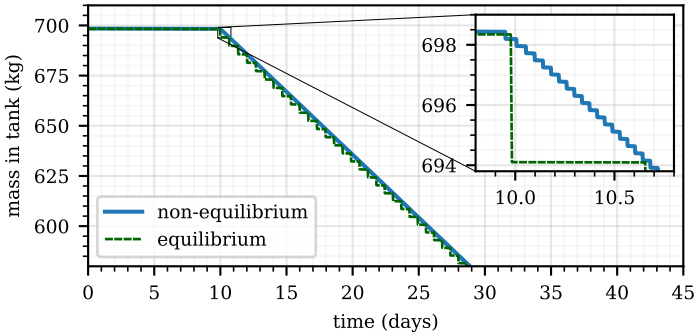


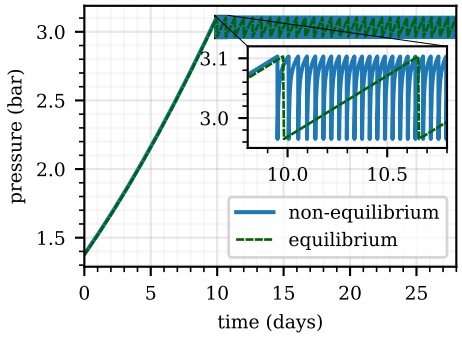
Fig. 3.1: Comparison of model to experimental [9] hydrogen release.

Figure 3.2(a) shows the hydrogen mass loss from the tank for the equilibrium and non-equilibrium models during the normal boil off. If the tank is left to vent under these conditions, it will empty over the course about 115 days. The pressure relief valve will safely handle the hydrogen release. Over the course of many releases, the equilibrium and non-equilibrium models are approximately the same. Each pressure relief episode in the non-equilibrium case releases approximately 0.25 kg of hydrogen, while in the equilibrium model, over 4 kg are released. In Figure 3.2(a), both the equilibrium and non-equilibrium models predict the hydrogen in the tank to heat up at a similar rate, illustrated by the time to start venting. The pressurization and heating are also illustrated in Figure 3.2. These results indicate that the evaporation rate between the liquid and vapor phases of the non-equilibrium model is not a bottleneck in pressurizing the tank. Heat transfer through the MLI is limiting the overall heat transfer, and drives the pressurization rate.

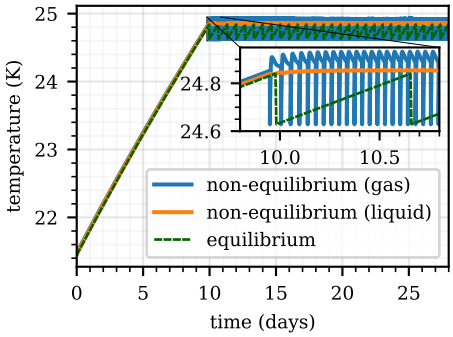
The major difference between the two models is how the venting cycles happen. For the equilibrium phase, when the vapor pressure drops from opening the pressure relief valve, some of the liquid phase immediately evaporates to preserve thermodynamic equilibrium. As a result, the pressure in the tank does not drop as fast, and more vapor leaves the tank before the 2.9 bar threshold to close the valve is reached. On the other hand, in the non-equilibrium case, when the vapor phase leaves the tank through the pressure relief valve and the vapor pressure drops, the dynamics of mass transfer become critical. The evaporation with the heat transfer through the MLI is not as fast as the instantaneous evaporation with the thermodynamic equilibrium, and as a result the vapor pressure drops to 2.9 bar quickly. However, since less energy is lost in each venting episode, less energy is required to pressurize the tank again and re-open the vent. This is also shown in the rapid temperature and pressure cycles in the tank for the non-equilibrium model in Figure 3.2(b) and Figure 3.2(c). In the non-equilibrium model, the liquid temperature is stable compared to the vapor temperature. The higher liquid temperature in the non-equilibrium case, compared to the equilibrium case, causes faster heat transfer to the vapor and faster pressurization.



(a) Mass loss



(b) Tank pressure



(c) Tank temperature

Fig. 3.2: Tank conditions for the non-equilibrium and equilibrium models with normal heat transfer.

4. Results. In this section, four liquid hydrogen storage tank pressure relief scenarios are explored: 1) Vacuum loss in the MLI layer, 2) An external fire engulfing the storage tank, 3) Loss of vacuum and an engulfing fire, and 4) High conduction through the insulation layer. The key metrics explored for the hydrogen releases are how fast hydrogen is released and whether or not the pressure relief valve is sufficient to lower the pressure of the tank to the rated tank pressure. All of the tank simulations in the results section have the same specifications as the tank described in Section 3 (Table 3.2), unless otherwise stated.

4.1. Loss of Vacuum. One quasi-real world scenario that these tanks may encounter is loss of vacuum in the insulation layer between the steel shells. The gap between the two steel shells is assumed to be filled with air, so the thermal conductivity, density, and heat capacity of air (κ_{air} = 0.022 W/m-K, ρ_{air} = 1.225 kg/m^3 $c_{p,air}$ = 1000 J/kg-K) were used for the insulation layer. The results are only shown for the non-equilibrium model.

Figure 4.1 shows that the pressure relief valve can safely reduce the tank pressure below 2.9 bar. With the loss of vacuum, the pressure relief valve begins to vent after approximately 2.4 hours and vents about 35 times an hour, compared to the normal heat transfer case where it starts to vent after nearly 10 days and at a frequency on the order of 1 time every hour. The rapid cycling of the valve may cause safety issues. The temperature of the liquid, vapor,

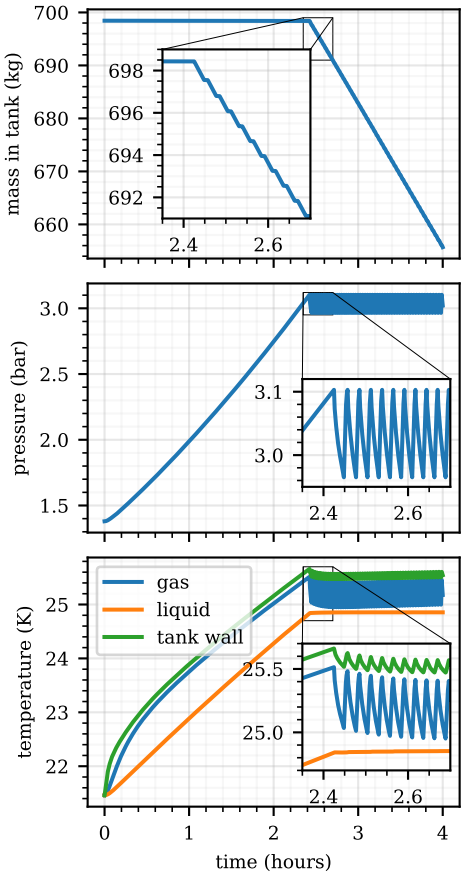


Fig. 4.1: Tank mass, pressure, and temperature for the case of a loss of vacuum in the insulation layer (κ_{air} = 0.022 W/m-K, ρ_{air} = 1.225 kg/m^3 $c_{p,air}$ = 1000 J/kg-K in the insulation layer).

and inside wall of the tank is shown in the bottom frame of Figure 4.1. The figure shows how close all of the temperatures are, and how the gas and wall temperatures both oscilate a bit as the relief valve opens and closes. The mass loss from the tank is shown in the top frame of Figure 4.1. If the pressure relief functions properly (as shown in the figure), it can safely reduce the pressure of the tank. The tank will release hydrogen continuously for about 25 hours at a rate of 27 kg/h. Each venting episode releases about 0.7 kg of hydrogen vapor.

4.2. Engulfing Fire. A second quasi-real world scenario that was explored was high external temperatures to the tank, possibly from a fire. The scenario considers the case where the ambient temperature around the hydrogen storage tank is 1200 K. Only convective heat transfer is considered, no radiation from the fire, and the entire ambient surroundings are at this temperature, not just one side of the tank. Only the non-equilibrium model is considered.

The plots for the case of the external fire are similar to the loss of vacuum in the

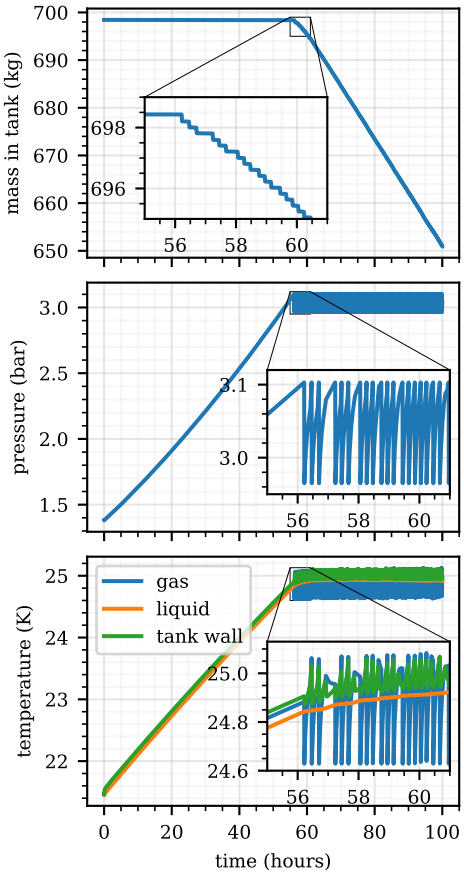


Fig. 4.2: Tank mass, pressure, and temperature for the case of an external fire (ambient temperature of 1200 K).

insulation. The middle frame of Figure 4.2 shows the pressure variation with the rapid opening and closing of the safety relief valve. The valve opens and closes about 6 times per hour. The bottom frame of Figure 4.2 shows the temperature cycling with the valve opening and closing. The reason the wall temperature changes so much is due to the low heat transfer through the MLI. Unlike the case of vacuum loss, where the heat transfer through the wall is significant, the heat transfer through the wall in this case is much lower. As a result, heat loss due to heat transfer to the vapor phase is not readily replaced with heat transfer through the MLI. The mass loss of hydrogen from the tank is shown in the top frame of Figure 4.2. The mass loss from the tank is managed with the pressure relief valve. The average venting rate is 1.1 kg/h, which indicates about 636 hours of venting to remove the hydrogen. Each venting episode release about 0.25 kg of gaseous hydrogen.

4.3. Engulfing Fire with Loss of Vacuum. The third examined case is the combination of the previous two cases, an external fire with loss of vacuum in the insulation layer. In other words the ambient temperature is set to 1200 K, and the thermal conductivity, density, and heat capacity of air ($\kappa_{air} = 0.022$ W/m-K, $\rho_{air} = 1.225$ kg/m^3 $c_{p,air} = 1000$ J/kg-K) were used in the middle layer of the wall.

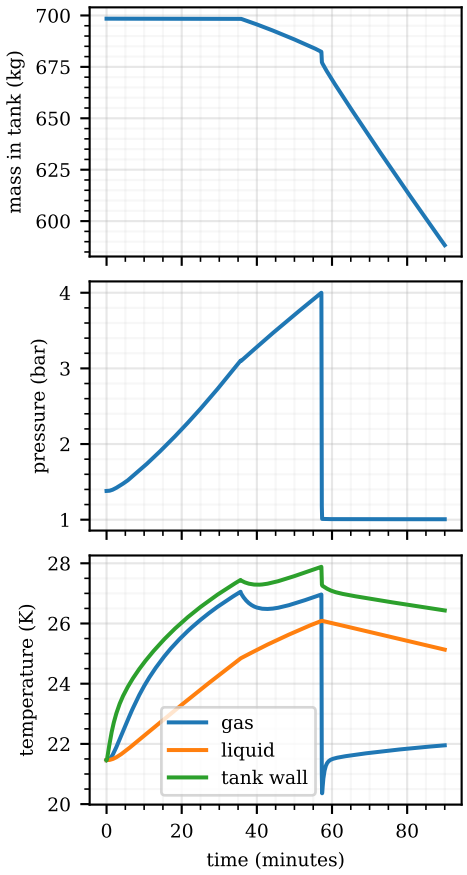


Fig. 4.3: Tank mass, pressure, and temperature for the case of an engulfing fire with a loss of vacuum in the insulation layer (ambient temperature of 1200 K, $\kappa_{air} = 0.022$ W/m-K, $\rho_{air} = 1.225$ kg/m^3 $c_{p,air} = 1000$ J/kg-K in the insulation layer).

The pressure and temperature shown in the bottom two frames of Figure 4.3 illustrate the details of what happens in the tank. As the tank heats up and pressurizes to 3.1 bar, the pressure relief valve opens, after 35 minutes. While this reduces the rate of pressurization, it does not reduce the pressure and the tank continues to pressurize to 4 bar, while still releasing mass. The pressure relief valve cannot handle this case. After the burst disc opens, the tank rapidly releases hydrogen until the vapor phase is at the pressure of the ambient environment. Then the tank continues to lose mass through the bust disc until it is empty. The mass loss for this scenario is shown in the top frame of Figure 4.3. The initial hydrogen release rate is about 0.7 kg/min when the pressure relief valve opens. When the burst disc opens, the hydrogen release is about 60 kg/min for 9 seconds, then it settles to a steady release rate of 2.7 kg/min.

4.4. High conduction through the insulation layer. To explore a hazardous hydrogen release scenario, a case with abnormally high conduction through the insulation layer is explored. In this scenario, the insulation thermal conductivity is set to 1 W/m-K.

Similar to the case of normal heat transfer to the tank, both the equilibrium and non-

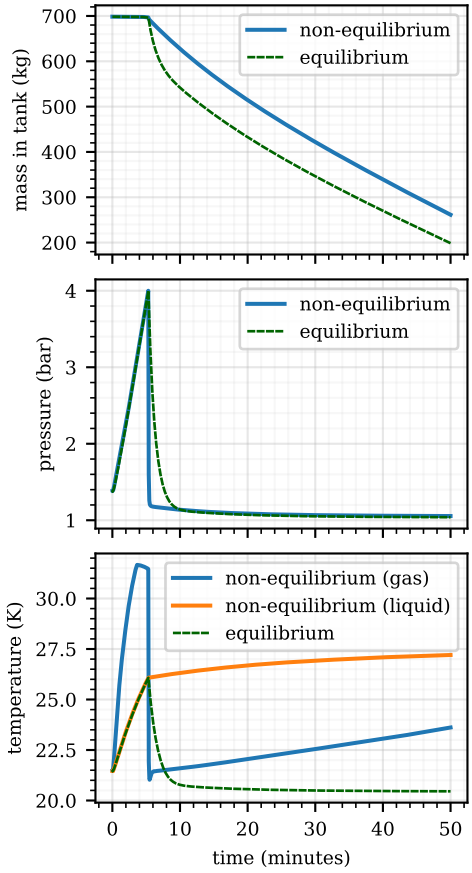


Fig. 4.4: Tank mass, pressure, and temperature for the case of increased conduction through the insulation layer ($\kappa_{\text{insulation}} = 1$ W/m-K).

equilibrium models start venting at the same time. This is shown in Figure 4.4. Interestingly, the bottom frame of Figure 4.4 shows how the vapor and liquid in the non-equilibrium tank are at significantly different temperatures. Since the vapor phases for both the equilibrium and non-equilibrium cases are at approximately the same pressure, the non-equilibrium gas must be at a lower density. The small dip in the equilibrium gas temperature, prior to the tank opening (3-5 minutes), is due to the liquid phase boiling. When the liquid boils, it adds saturated vapor to the vapor phase, which cools down the vapor.

The pressure relief valve opens at 3.1 bar, but does not have a noticeable impact on depressurizing the tank as seen in the middle frame of Figure 4.4. Instead the tank pressure rises to 4 bar and the burst disc opens. Again, the venting shows major differences between the equilibrium and non-equilibrium models. In the equilibrium model, the pressure drop is slower, but there is a more dramatic loss of mass from the tank initially. In the non-equilibrium model the vapor phase loses pressure very quickly. The mass loss is then limited by evaporation and boiling from the liquid phase, which quickly leaves through the burst disc. Finally, both the equilibrium and non-equilibrium models reach a state where the mass loss is limited by heat transfer through the tank, at around 10 minutes, where the pressure

in both models is around atmospheric.

The mass loss from this tank is illustrated in the top frame of Figure 4.4. In this case the pressure relief valve cannot safely manage the tank pressure. The mass is lost from the tank over the span of about an hour. In the non-equilibrium case, the first 100 kg are vented from the tank in about 6 minutes, while the equilibrium model predicts this mass loss to occur in under a minute. After this initial high rate of venting, both releases eventually settle to a mass loss rate of about 9 kg/min.

5. Conclusions. Five scenarios for a hydrogen storage tank were examined. First, a validation scenario with normal boil-off was compared to experimental data from a real world tank. Then four abnormal scenarios were explored, a loss of vacuum in the insulation layer, a high ambient temperature (to simulate an engulfing fire), a high ambient temperature with a simultaneous loss of vacuum, and high conduction through the insulation layer. Only the cases with extreme heat transfer to the tank caused the burst disc of the hydrogen tank to open and release hydrogen quickly (30-60 kg/min). In the case of loss of vacuum or external heat from a fire, the pressure relief on the tank managed to keep the pressure on the tank down. The result was a controlled release of hydrogen from the tank and at a much faster rate than the normal heat transfer.

Each venting episode through pressure relief valve releases hydrogen at the same rate (0.7 kg/min). The release is a function of the valve and not the specific heat transfer cases. Changing the valve specifications would change that rate. When the pressure relief valve suffices to control the hydrogen release, the valve opening frequency depends on the rate of heat transfer to the tank, higher heat transfer leads to more frequent valve openings, and the volume of liquid in the tank changes the duration of the release, less liquid (more vapor) leads to a longer release. The average mass loss form the tank is proportional to the heat transfer to the tank.

When the pressure relief valve is sufficient to control the release, and many release events occur, then the equilibrium model works as a good surrogate to the non-equilibrium model. However, when only a single venting event occurs, abnormal transfer or fire and vacuum loss, the equilibrium model does not work well. In those cases the mass transfer between the liquid and vapor phase (evaporation and boiling) become critical. The boiling mass transfer was a critical addition to avoid simulations where the liquid was substantially hotter than the vapor.

Each scenario reveals the different hydrogen releases that a hydrogen dispensing station may experience. Those releases can be taken a step further to model flames or hydrogen mass transfer to help set safety codes and standards for hydrogen storage tanks. Those standards can help make stations safe and be built on the smallest footprint.

5.1. Future work. It is clear from Section 3 that the experimental release of hydrogen from the storage tanks decrease with tank volume. However, the presented model has a constant release of hydrogen regardless of tank fill. Since the mass loss in the tank is so closely tied to heat transfer, it is likely that the heat transfer through the tank depends on the height of liquid in the tank. One possible explanation is that the MLI properties are impacted by the height of liquid in the tank. Another explanation is that the model does not consider a 2-D model for the wall temperature, which may cause differences in heat transfer through the tank.

Other potential improvements to the model include the use of subcooled vapor or superheated liquid states. However that would require a departure from CoolProp. Temperature dependent wall properties (C_p and κ) may improve accuracy. Finally adding spatial discretizations to the vapor and liquid would add granularity to the simulation. The current 0-D model assumes instantaneous mixing of the hydrogen. As a result, when boiling occurs

in the model ($T_w - T_L > 0.1$), rapid heat transfer happens in part because the heat transfer rate through the liquid is assumed to be instantaneous. In other words, the current boiling heat transfer model overestimates the heat transfer from the wall to the liquid. A correction factor or maximum heat transfer rate could be considered since the cooler liquid at the middle of the liquid phase needs time to reach the wall. Instead of boiling the system could also be modeled as flash evaporation, like the work of Tani et al. [11].

Experiments to more precisely determine the mass transfer mechanism between the liquid and vapor phase would help with modeling. In the hazardous release scenarios, that is a critical value that keeps the hydrogen from releasing as much hydrogen as predicted by the non-equilibrium model. In order to assist in the validation and elucidation of these mechanisms, detailed specifications of the tank are needed.

REFERENCES

[1] I. H. Bell, J. Wronski, S. Quoilin, and V. Lemort, *Pure and pseudo-pure fluid thermophysical property evaluation and the open-source thermophysical property library coolprop*, Industrial & Engineering Chemistry Research, 53 (2014), pp. 2498–2508.

[2] R. Bozinoski, *Masstran (v0.19.1) theory guide*, Tech. Rep. SAND2019-7163, Sandia National Laboratories, 2019.

[3] S. W. Churchill and H. H. Chu, *Correlating equations for laminar and turbulent free convection from a vertical plate*, International journal of heat and mass transfer, 18 (1975), pp. 1323–1329.

[4] M. Daigle, M. Foygel, and V. Smelyanskiy, *Model-based diagnostics for propellant loading systems*, in 2011 Aerospace Conference, IEEE, 2011, pp. 1–11.

[5] P. N. Estey, D. H. Lewis Jr, and M. Connor, *Prediction of a propellant tank pressure history using state space methods*, Journal of Spacecraft and Rockets, 20 (1983), pp. 49–54.

[6] F. P. Incropera, A. S. Lavine, T. L. Bergman, and D. P. DeWitt, *Principles of heat and mass transfer*, Wiley, 2013.

[7] V. Osipov and C. Muratov, *Dynamic condensation blocking in cryogenic refueling*, Applied Physics Letters, 93 (2008), p. 224105.

[8] V. V. Osipov, M. J. Daigle, C. B. Muratov, M. Foygel, V. N. Smelyanskiy, and M. D. Watson, *Dynamical model of rocket propellant loading with liquid hydrogen*, Journal of Spacecraft and Rockets, 48 (2011), pp. 987–998.

[9] G. Petitpas, *Simulation of boil-off losses during transfer at a lh2 based hydrogen refueling station*, International Journal of Hydrogen Energy, 43 (2018), pp. 21451–21463.

[10] P. Sutheesh and A. Chollackal, *Thermal performance of multilayer insulation: a review*, in IOP Conference Series: Materials Science and Engineering, vol. 396, IOP Publishing, 2018, p. 012061.

[11] K. Tani, T. Himeno, Y. Sakuma, T. Watanabe, H. Kobayashi, T. Toge, H. Kagaya, S. Kamiya, and O. Muragishi, *Prediction of pressure reduction rate in 30 m3 liquid hydrogen tank based on experimental and numerical analysis*, in Proceedings of the International Conference on Hydrogen Safety, 2019.

[12] L. Wang, Y. Li, F. Zhang, F. Xie, and Y. Ma, *Correlations for calculating heat transfer of hydrogen pool boiling*, International Journal of Hydrogen Energy, 41 (2016), pp. 17118–17131.

Appendix A. Code.

The simulations were run using MassTran 0.19.3, Python 2.7, and with CoolProp 6.3.0. There is work in progress to make MassTran compatible with Python 3.

A.1. Example input file.

```
# System definition
System:
  source:
    type: NonEqTank
    eos_init: #this is the liquid
        quality: 0
        pres: 138000.0
        #temp: 15
```

```
        mole_fractions:
            Hydrogen: 1.0
    eos_init_vapor: #this is the vapor
        quality: 1
        pres: 138000.0
        #temp: 30.0
        mole_fractions:
            Hydrogen: 1.0
    geometry_init:
      shape: 'cylinder'
      length: 3.97
      diameter: 2
    fraction_liquid: 0.8
    wall_heat_transfer:
        temperature_amb: 300
        heat_transfer_coef_amb: 10 #w/m2-K
        init_wall_temp: [21.46, 21.46, 21.46, 21.46, 21.46, 93, 162,
            ↪ 231.3, 300, 300, 300, 300, 300] #estimate initial wall
            ↪ temps. finite volumes = sum numxl (15) - #walls (3) + 1
        wall_properties:
            wall_1: #steel
                wall_thickness: 0.0111  # From Petitpas
                rho_wall: 8050.0 # kg/m^3
                cp_wall: 25  # J/kg-K
                thermal_conductivity_wall: 3 #W/m-K #204.2
                numxl: 5 #must be at least 3, code assumes a middle layer
                wall_2: #insulation/air
                wall_thickness: 0.0508  #m
                rho_wall: 0.1 #kg/m^3  #0.1
                cp_wall: 0.1  #J/kg-K  #0.1
                thermal_conductivity_wall: 20.0E-5 #1 #20.0E-5 #1 #0.022
                    ↪ #W/m-K
                numxl: 5
            wall_3: #steel
                wall_thickness: 0.0038  #m
                rho_wall: 8050.0 # kg/m^3
                cp_wall: 450  # J/kg-K
                thermal_conductivity_wall: 15 #W/m-K  #204.2
                numxl: 5 #must be at least 3, code assumes a middle layer

relief_valve:
   type: PressureValve
   geometry_init:
     diameter: 0.005  # From Petitpas
   upstream: source
   downstream: sink
   init_open: False
   atol:
     pressure: 1.0e-6
   close_pres_ratio: 2.965   #43 psi
   open_pres_ratio: 3.103 #45 psi

burst_orifice:
   type: BurstOrifice
```

```
    geometry_init:
      diameter: 0.038  #1.5"
    upstream: source
    downstream: sink
    open_pres_ratio: 4.0 #Pressure ratio between the upstream and
        ↪ downsream where the orifice opens and stays open

  sink:
    type: BoundaryNode
    eos_init:
        temp: 300.0
        pres: 1.e+5
        mole_fractions:
            Hydrogen: 1.0
    geometry_init:
      shape: 'sphere'
      volume: 1.0e+20
    wall_heat_transfer:
        model: combined
        wall_temperature: 300.0
    atol:
        pressure: 1.0e-6
        temperature: 1.0e-6

Solver:
  simulation_title: "Non-equilibrium Hydrogen Tank"
  sim_time: 2592000
  atol: 1.0e-4
  rtol: 1.0e-4
  #maxh: 1000 #maxh designates the maximum time step the solver can take

Fluid:
    species:
        - Hydrogen
Output:
    format: csv #hdf5 #csv
    file_name: Non_Eq_normal
```

A.2. Timing and tolerances. The equilibrium model runs about 100 times faster than the non-equilibrium model. There are two main reasons for this. The first is that the valve opens/closes far more often in the non-equilibrium model than in the equilibrium model. Those state events effectively cause the simulation to pause and to restart. The second is IF statements in the non-equilibrium code that cause discontinuities and therefore smaller time steps. The culprits are the boiling and condensation IF statements. The disparity between the equilibrium and non-equilibrium model is especially large when those items are triggered. The issue was that the when evaporation occurred (the liquid was saturated and nucleate boiling was possible) the liquid would jump to being just below saturation and oscillate between those two regimes. The heat transfer with boiling is significantly different than the convective heat transfer. This may be rectified with a less aggressive boiling heat transfer regime.

One work around for the boiling model was to let the vapor become slight superheated and then instead of using the heat transfer from the wall dictate the amount of boiling, the difference between the enthalpy of the superheated liquid and the enthalpy of the saturated

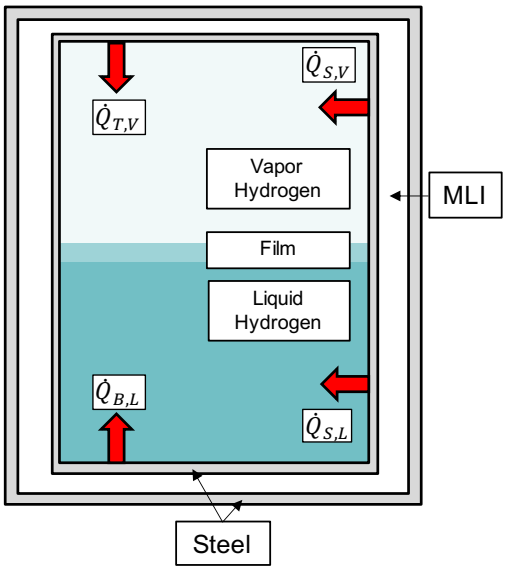


Fig. A.1: Four heat transfer locations from the wall to the hydrogen.

liquid was the driving force for evapoaration, modified with a constant. This worked well except for the case of large pressure drops where the enthalpy of saturated liquid also dropped drastically. Therefore the minimum of those two evaporation heat transfer regimes was taken, which is mathematically presented in A.1

$$\dot{m}_{boil} = \min\left(\frac{K(H_L - H_{L,sat})}{H_{vap}}, \frac{\dot{Q}_{WL}}{H_{vap}}\right) \tag{A.1}$$

A value of K = 100 worked well to mimic the wall boiling results.

For the abnormal heat regime tolerances at 10^{-4} worked well in the trade off accuracy and run time. Still, those simulations took 10-15 minutes.

A.3. Implicit wall heat transfer. The implementation of implicit wall heat transfer was built into both the equilibrium and non-equilibrium tank node components for hydrogen transportation of MassTran. The addition of the residual equations allowed for faster computation since it could handle larger time steps than the explicit equations. Figure 2.1 shows the discretization of the wall into finite volumes. The code has pre-built discretizations for cylinder and sphere volumes, which is automatically selected based on the geometry of the system. In the discretization, x represents the distance from the inside of the tank through the shell, V represents the wall finite volume, A represents the interface area.

The properties of the wall, heat capacity (C_p), thermal conductivity (κ), and density (ρ), are constant. Density and heat capacity are applied to the volume centers. Heat capacity is applied to the interfaces. When two different layers touch (e.g., MLI and Steel), a cell made out of half of each layer will be created, as seen in Figure 2.1, with the average density and heat capacity of the two layers. The interface touching the MLI with have the thermal conductivity of MLI and same for the steel.

Three types of wall heat transfer occur. For notation of the finite volume, i represents cell counts starting at the inside most wall volume (i=1) to the outside most wall volume

(i=n). Using the orientation of Figure 2.1, there is convection from the hydrogen to the wall from the left and conduction to the next wall component to the right (i=1), interior conduction (i=2:n-1), and conduction from the left and convection to the right to ambient conditions at (i=n).

The residual equation for i=1 is shown in (A.2)

$$res = -\rho_i C_{p,i} V_i \frac{dT_i}{dt} - \dot{Q}_{L,W} - \dot{Q}_{V,W} + A_i \kappa_i \frac{T_{i+1} - T_i}{x_{i+1} - x_i} \tag{A.2}$$

The residual equation for the interior volumes i=2:n-1 is shown in (A.3)

$$res = -\rho_i C_{p,i} V_i \frac{dT_i}{dt} - A_{i-1} \kappa_{i-1} \frac{T_i - T_{i-1}}{x_i - x_{i-1}} + A_i \kappa_i \frac{T_{i+1} - T_i}{x_{i+1} - x_i} \tag{A.3}$$

The residual equation for the exterior volumes i=n is shown in (A.4)

$$res = -\rho_i C_{p,i} V_i \frac{dT_i}{dt} - A_{i-1} \kappa_{i-1} \frac{T_i - T_{i-1}}{x_i - x_{i-1}} - A_i h_{amb} (T_{amb} - T_i) \tag{A.4}$$

The derivatives of wall temperatures can be large at the beginning of simulations, which can cause it to crash. Therefore, careful thought should go into the those temperature distributions. In the input file, the number of layers (e.g., Steel, MLI, Steel) and the number of discretization of those layers with numxl can be selected. As mentioned above, the interface of two layers will merge together to form a joint finite volume, thereby reducing the number of finite volumes by 1. The result is that 3 layers, each discretized into 5 volumes, will have 13 finite volumes total ($3 \times 5 - 2$ interfaces).

When these equations are entered into MassTran formatting, the T values are replaced with y[index]. y[index] tells the solver where the specific T value is stored in the solver. Likewise, the $\frac{dT}{dt}$ values are replaced with yd[index].


CSRI
SUMMER
PROGRAM 2020